Kollaborative Visualisierung großer, interaktiver und dynamischer 3D Szenen auf verteilten Endgeräten

Vom Fachbereich Informatik (FB 20)
der Technischen Universität Darmstadt genehmigte

Dissertation

zur Erlangung des akademischen Grades
eines Doktor–Ingenieurs (Dr.–Ing.)

von

Dipl.–Inf. Jörg Sahm

geboren in Offenbach

Referent:	Prof. Dr.–Ing. Dr. h.c. Dr. E.h. José L. Encarnação, Technische Universität Darmstadt
Korreferent:	Prof. Dr. Reinhard Klein, Universität Bonn
Tag der Einreichung:	14.05.2004
Tag der mündlichen Prüfung:	25.06.2004

Darmstädter Dissertation
D 17
Darmstadt, 2004

Widmung

Allen, die ich liebe

Vorwort

Die vorliegende Dissertation entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Fraunhofer Institut für Graphische Datenverarbeitung (IGD) in Darmstadt sowie als wissenschaftlicher Mitarbeiter der TU Darmstadt im Bereich Graphisch Interaktive Systeme (GRIS).

Mein besonderer Dank gilt den zahlreichen Personen und Institutionen, die auf unterschiedliche Art und Weise zum Gelingen der Arbeit beigetragen haben. Dazu zählen

- Herr Prof. Dr.–Ing. Dr. h.c. Dr. E.h. José L. Encarnação, dem ich für die Annahme und Betreuung dieser Dissertation, für die gemeinsame Durchführung von Diplomarbeiten und der nicht ganz unwesentlichen Bereitstellung meines Arbeitsplatzes danke,

- Herr Prof. Dr. Reinhard Klein, dem ich für sein Interesse und die Bereitschaft zur Übernahme des Koreferats danke,

- Herr Dr. Nasko sowie die gesamte Heinz Nixdorf Stiftung, denen ich für ihr Interesse sowie die Ermöglichung des Projekts Ubiquitous Graphical Objects danke,

- Herr Dr. Volker Luckas, dem ich in seiner Funktion als Abteilungsleiter für sein Vertrauen in meine Arbeit und der langjährigen Zusammenarbeit danke,

- meine Kollegen in der Abteilung Animation und Bildkommunikation (A3), insbesondere Eric Blechschmitt, Arnulph Fuhrmann, Clemens Gross, Markus Hoffmann, Thorsten Mai, Sascha Schneider, Ingo Soetebier, die mir viele Anregungen und Tipps gegeben haben und denen ich für die sehr gute Zusammenarbeit danke,

- mein ehemaliger Kollege und Projektmitarbeiter Horst Birthelmer, der mir mit seinen Ideen und seiner Erfahrung viel geholfen hat. Wir waren zusammen mit Ingo ein Spitzenteam und werden es hoffentlich irgendwann auch wieder sein,

- meine Kollegen im Institut, insbesondere Gabi Knöß, Carola Eichel, Matthias Finke und Norbert Braun, denen ich für ihre Unterstützung bei meiner Arbeit danke,

- meine Diplom-, Studien- und Bachelorarbeitern, Praktikanten sowie meine wissenschaftlichen Hilfskräfte Stefan Barthel, Stefan Böhm, Andreas Heizenreder, Kirstin Saufaus und Sven Schäfer, die mir ebenfalls viele Anregungen gegeben haben,

- meine Freunde zu Hause und Umgebung, denen ich für ihre Ablenkung und die gemeinsame Freizeit danke,
- meiner Schwester Astrid, der ich für die moralische Unterstützung danke,
- und vor allem meine Eltern Gisela und Heinz, für die ich immer da sein werde.

Darmstadt im Mai 2004 Jörg Sahm

Inhaltsverzeichnis

Kapitel 1

Einleitung

Durch die rasante Entwicklung der letzten Jahre wird der Mensch immer öfter und frühzeitiger mit dem Computer konfrontiert. Seien es automatische Kinokartenbestellsysteme, komfortable Faxgeräte oder gar Geräte für den normalen Haushalt, fast überall verrichten kleine Mikrochips ihre Arbeit mit einer Leistungsfähigkeit, für die noch vor wenigen Jahren aufwendige Anlagen vonnöten waren. Der Umgang mit diversen Ein- und Ausgabegeräten wie Tastaturen und grafischen Anzeigen ist dabei zu einer Selbstverständlichkeit geworden.

Mit zunehmendem Maße hält der Computer auch in den Kinderstuben seinen Einzug. So ist die Informatik mittlerweile ein anerkanntes Unterrichtsfach an den Schulen. Spezielle Lernprogramme wie beispielsweise die Agentsheets [Age] in den USA, unterstützt von der National Science Foundation, sollen den Kindern schon im frühen Alter das Programmieren näher bringen. Da der Mensch die Auswirkungen der elektronischen Datenverarbeitung über die verschiedenen Sinneswahrnehmungen direkt konsumiert, ist der Einfluss der Medien ebenfalls nicht zu vernachlässigen. Wurden früher kurze Zeichentrickfilme über enormen Aufwand von Hand erstellt, so sind seit Anfang der neunziger Jahre die ersten vollständig computeranimierten Kinofilme zu bewundern. Beginnend mit Toy Story bis hin zu Shrek und Final Fantasy erwecken die präsentierten Bilder inzwischen einen derart realistischen Eindruck, dass der Gedanke, Schauspieler durch gescannte 3D Profile zu ersetzen, nicht länger eine Illusion bleiben muss.

Das Beispiel der Kinofilme führt im wahrsten Sinne des Wortes die wichtige Fähigkeit des Computers vor Augen, Informationen optisch darzustellen zu können. Hierbei gewinnt nicht nur die Anzeige von einzelnen Bildern weiterhin an Bedeutung, sondern im besonderen die Wiedergabe von bewegten Animationen, welche unter anderem im Bereich der Werbung nicht mehr wegzudenken sind. Der Eindruck der Dynamik einer Animation entsteht aus einer Abfolge rasch aufeinander eingeblendeter Einzelbilder mit jeweils geringen Änderungen. Das Erstellen dieser Einzelbilder ist zum einen immer noch mit enormem Aufwand und zum anderen mit künstlerischem Können verbunden. Deshalb entwickelt sich der Trend dahingehend, anstelle zweidimensionaler Bilder die gesamten Kulissen als 3D Szenen in digitaler Form zu modellieren. Die Animationen selbst werden durch Kamerafahrten oder dynamische Elemente innerhalb der Szenen erzeugt. Das Positionieren von Kameras und Lichtquellen erlaubt die automatische Berechnung von realitätsnahen Bildern, ohne dass sich der Benutzer

mit Aspekten wie Perspektive und Schattenwürfen auseinander setzen muss.

Durch die enorm gestiegene Leistungsfähigkeit der Computer ist es mittlerweile möglich, derartige 3D Szenen in beachtlicher Geschwindigkeit und Genauigkeit zu präsentieren. Der Wunsch nach immer detaillierteren und realistischeren Bildern steht dieser Entwicklung jedoch entgegen, da aufgrund der hiermit wachsenden Datenmengen und der Entwicklung neuer Techniken wie 3D Texturen oder Lichtberechnungen die Grenzen der Leistungsfähigkeit spielend erreicht werden.

1.1 Motivation

In den letzten Jahren ist mit dem Internet ein neues wichtiges Medium entstanden, welches nun einem breiten Publikum zur Verfügung steht. Entsprechend dem Ressourcenspektrum der Computer ist auch hier die Entwicklung nicht stehen geblieben. Netztopologie, Bandbreite und Verfügbarkeit haben sich enorm verbessert, so dass inzwischen die Übertragung von visuellen Informationen möglich ist. Im Gegensatz zu den kurzen und sicheren Übertragungswegen zwischen Prozessor, Speicher und Grafikkarte ist die Leistungsfähigkeit in derzeitigen Netzwerken allerdings noch stark eingeschränkt. Geringe Bandbreiten, Übertragungsfehler und die Anfälligkeit der Leitungen gegenüber nicht autorisierten Zugriffen stellen immer noch ein schwerwiegendes Problem dar. Dennoch wird im Bereich der Produktwerbung bereits umfassend von den Möglichkeiten des Internets Gebrauch gemacht und einzelne Artikel in Form einfacher 3D Modelle präsentiert.

Mit der Verbreitung des Internet und dem Wunsch nach einer immer flexibleren Kommunikationsfähigkeit ist auch der Bedarf an der permanenten Erreichbarkeit gewachsen. Geschäftsleute, deren Erfolg eng mit einer schnellen Kenntnisnahme von kosmopolitischen Ereignissen und der sofortigen Reaktion auf diese Situationen verwoben ist, sind auf stets verfügbare Kommunikationsmittel angewiesen. Begriffe wie E-Commerce, E-Business und Home Banking hielten ihren Einzug. Der normale Personal Computer erwies sich hierfür als unzureichend, da er aufgrund von Größe und Gewicht nicht mobil einsetzbar war. Aus diesem Grund wurde eine Vielzahl neuer Geräte wie Laptops, Palmtops und Handys entwickelt, deren Leistungsfähigkeit sich zwar rasant steigert, dem Stand der aktuellen Personal Computer allerdings nachsteht.

Zwischen den einzelnen Geräten existieren verschiedene Schnittstellen, über die ein Austausch von Daten erfolgen kann. In der Regel handelt es sich dabei um die Übergabe von textuellen Informationen wie Terminen und Nachrichten sowie im eingeschränkten Maße um das Streamen von Video- und Soundsequenzen. Das Übertragen von dreidimensionalen Geometriedaten wird derzeit lediglich für sehr einfache 3D Modelle zwischen Personal Computern und den ähnlich leistungsfähigen Laptops eingesetzt, wodurch einerseits wichtige Anwendungen und andererseits eine Vielzahl ungelöster Probleme ignoriert werden. Der folgende Abschnitt skizziert denkbare Anwendungen und damit verbundene Probleme.

1.2 Problemstellung und Zielsetzung

Wird der Gedanke an die Übertragung von dreidimensionalen Geometriedaten um die Visualisierung dieser Informationen auf verschiedenen verteilten Endgeräten erweitert, könnten beispielsweise Bauplaner von beliebigen Standorten aus Architekturskizzen anfordern, betrachten, gemeinsam editieren und kommunizieren. Gleiches wäre auch bei dem Design von einfachen Artikeln bis hin zu komplexen Automobilen und anderen Fahrzeugen denkbar. Ein Manager könnte im Flugzeug sitzend den aktuellen Entwurf eines neuen Produktes per Internet laden und begutachten. Verfügt sein Endgerät über ein Mindestmaß an Leistungs- und Interaktionsfähigkeit wäre er sogar imstande, den Entwurf zu verändern und als Korrektur zurückzusenden.

Handelt es sich bei der Übertragung und Darstellung von Skizzen und Entwürfen noch um statische Szenen, so ändert sich dies schlagartig bei der Visualisierung von Zuständen, welche sich über die Zeit verändern können. Hieraus resultieren weitere wichtige Anwendungsfälle wie die Präsentation von dynamischen Arbeitsvorgängen oder Simulationsdaten. Mehrere Gutachter könnten von verschiedenen Standorten aus einen Vorgang beobachten und beurteilen. Dürfen die Positionen der Gutachter zusätzlich einem mobilen Kontext unterliegen, so wäre auch eine Beurteilung auf Reisen möglich.

Da der notwendige Detaillierungsgrad bei Arbeitsvorgängen und Simulationen häufig sehr hoch ist, ergibt sich aus diesen Anwendungen der Bedarf nach der Darstellung von großen, interaktiven und dynamischen 3D Szenen auf verschiedenen verteilten Endgeräten. Interaktivität bedeutet hierbei nicht nur die Veränderung des Beobachterstandpunktes sowie der Blickrichtung, sondern auch die Kommunikation der Szenenelemente mit- und untereinander. Darüber wäre zum Beispiel das Erkennen von Kollisionen möglich. Die Dynamik der Szenen basiert auf eventuellen Zustandsänderungen der Szenenelemente wie Position, Größe und Orientierung.

Bei der Visualisierung dreidimensionaler Szenen sind die unterschiedlichen Leistungsmerkmale der jeweiligen Endgeräte zu berücksichtigen, welche unter anderem in Speicher, Auflösung und Rechenleistung variieren. Soll die gleiche Szene über verschiedene Endgeräte betrachtet werden, so müssen bestimmte Anpassungen der Datenmengen und der Daten selbst erfolgen. Beispielsweise ist es derzeit nicht sinnvoll, Texturen auf einem Palmtop darzustellen. Auch die beschränkte Speicherkapazität muss berücksichtigt werden, da komplexe 3D Szenen in Dateien abgelegt häufig mehrere Gigabytes beanspruchen. Um derartige Files überhaupt in den Hauptspeicher eines Endgerätes laden zu können, müssen die eingelesenen Daten zum einen reduziert und zum anderen optimal selektiert werden. Eine Reduzierung von 3D Modellen kann über Simplifizierungsverfahren erfolgen, wobei aber zu berücksichtigen ist, dass die Semantik der Modelle erhalten bleibt. So sollte ein Auto auch nach der Simplifizierung als solches zu erkennen sein. Die optimale Selektion der Informationen ergibt sich aus dem, was der Benutzer von einer Szene betrachten möchte. In der Regel begutachtet der Anwender nicht eine komplette Szene, sondern lediglich einen bestimmten Ausschnitt. Da das Laden der riesigen Dateien einen enormen Zeitraum beansprucht, muss es möglich sein, in Abhängigkeit von Betrachterstandpunkt und Blickrichtung einen Teil der Szene schnell zu identifizieren, einzulesen und darzustellen. Dies impliziert eine effiziente Navigation innerhalb der Szene

und der damit verbundenen Datenmengen.

Aufgrund der Vielzahl der möglichen Verbindungen über Funknetze bis hin zu Local Area Networks (LAN) sind die Leistungsmerkmale nicht nur auf die Endgeräte beschränkt. Hier ist besonders die Eigenschaft der Bandbreite zu berücksichtigen, welche den Datendurchsatz pro Sekunde beschreibt. Es wäre fatal, eine komplexe 3D Szene über ein Funknetz mit einer geringen Bandbreite zu übertragen. Im Gegensatz zu Auflösung, Speicher und Rechenleistung handelt es sich bei der Bandbreite zudem um eine äußerst dynamische Eigenschaft, die enormen Schwankungen unterliegen kann. Somit müssen vor der Übertragung einer Szene auf ein Endgerät nicht nur dessen Kapazitäten sondern auch die Eigenschaften des Übertragungskanals in die Reduzierung und Selektion der Daten eingebracht werden. Dieser Umstand erfordert zum einen die stetige Überwachung der Kapazitäten, zum anderen aber auch die dynamische Adaption von Reduktion und Selektion zur Laufzeit.

Ein weiteres Problem bei dem Transfer von 3D Szenen rekrutiert aus dem Verlust von Informationen aufgrund von Übertragungsfehlern. In der Regel stehen die übertragenen Informationen in wechselseitigen Beziehungen. Geht nur eine einzige Information verloren, so werden die darauf folgenden Daten falsch interpretiert und es kommt zu massiven Darstellungsfehlern. Aus diesem Grund ist eine Quality of Service erforderlich, welche dem Benutzer die Garantie für eine fehlerredundante Übertragung seiner Daten zusichert.

Interaktive Szenen bringen nicht nur das Problem der Kommunikation zwischen den Elementen einer Szene mit sich, sondern auch das der Kommunikation der Benutzer mit Elementen und untereinander. Navigieren zum Beispiel mehrere verteilte Benutzer in der gleichen 3D Szene, so können sämtliche Anwender mit demselben Element innerhalb der Szene interagieren. Da Veränderungen aufgrund der Latenzzeiten der Übertragungskanäle erst nach einem bestimmten Zeitraum für die übrigen Benutzer sichtbar werden, können sich die Eingaben kumulieren, neutralisieren oder blockieren. Hieraus resultiert der Bedarf nach einer Synchronisation für den Zugriff auf die Datenmenge der Szene.

Auch mit einer geeigneten Synchronisation kann es bedingt durch die verschiedenen unterstützten Eingabemedien der jeweiligen Endgeräte zu unerwünschten Effekten kommen. So bieten Workstations in der Regel einen höheren Eingabekomfort als beispielsweise Palmtops, weshalb ein Benutzer mit einer Workstation wesentlich effektiver Elemente einer Szene manipulieren könnte als ein mobiler Benutzer. Aus diesem Grund muss ein Interaktionsmodell entwickelt werden, welches für den Benutzer ein möglichst intuitives und einheitliches Navigieren und Editieren der Szene auf sämtlichen Endgeräten erlaubt.

In den folgenden Kapiteln soll nun ein Konzept vorgestellt werden, welches die oben aufgeführten Probleme berücksichtigt. Grundlegendes Ziel ist die gleichzeitige Darstellung einer großen, dynamischen und interaktiven 3D Szene auf mehreren Endgeräten, welche über ein Netzwerk verbunden sind. Weitere Ziele stellen die dynamische Reduzierung und Selektion der Daten bedingt durch die parametrisierbaren Eigenschaften von Endgerät und Übertragungskanal, die Synchronisation der Benutzerinteraktionen mit Elementen der Szenen und anderen Benutzern sowie die intuitive und einheitliche Navigation auf den verschiedenen Endgeräten dar.

1.3 Gliederung

In der vorliegenden Arbeit wird ein Konzept zur kollaborativen Visualisierung großer, interaktiver und dynamischer 3D Szenen auf verteilten Endgeräten beschrieben. Die Aufzählung am Ende dieses Abschnitts gibt einen Überblick auf Inhalt und Gliederung der einzelnen Kapitel. Der Bezug zu existierenden Ansätzen und Konzepten, die sich mit der hier behandelten Problematik beschäftigen, ist dreigeteilt in den Kapiteln 3-5 zu finden: Kapitel 3 beschreibt nicht nur potentielle Anwendungsgebiete, sondern analysiert auch Branchen, in denen Techniken verteilter 3D Grafik bereits verwendet werden. Kapitel 4 stellt neben der Definition von Begrifflichkeiten die derzeit wichtigsten Ansätze dieser Techniken vor, allerdings ohne eine Wertung für deren Verwendung bei einer Realisierung der gesetzten Ziele zu geben. Letzteres ist Teil von Kapitel 5, welches detaillierte Anforderungen für die einzelnen Technologien vorgibt und den Bedarf nach neuen Ansätzen und Konzepten klärt.

- **Kapitel 2 (Anforderungen):** Dieses Kapitel formuliert die grundsätzlichen Ziele und Anforderungen der Arbeit, ohne sich in den technischen Details verschiedener Technologien zu verlieren.
- **Kapitel 3 (Anwendungsgebiete und Einsatzmöglichkeiten):** Hier wird der Frage nachgegangen, welche Anwendungen und Einsatzmöglichkeiten sich bei einer erfolgreichen Umsetzung der Anforderungen ergeben. Da bereits Bereiche existieren, in denen verteilte 3D Grafiken zur Geltung kommen, werden diese im Sinne der zuvor aufgestellten Anforderungen analysiert.
- **Kapitel 4 (Grundlagen):** In den Grundlagen werden die für das Umsetzen der Anforderungen notwendigen Technologien vorgestellt und entsprechende Begriffe eingeführt. Dabei erfolgt auch eine kurze Beschreibung wichtiger Veröffentlichungen, die jedoch lediglich im Rahmen grundsätzlicher Problematiken bewertet werden.
- **Kapitel 5 (Wissenschaftliche Einordnung):** Im Gegensatz zu Kapitel 2 geht dieser Teil der Arbeit auf die notwendigen technischen Aspekte für eine Realisierung der aufgestellten Ziele ein. Anhand der daraus aufgestellten Anforderungen analysiert Kapitel 5 existierende Vorgehensweisen und erläutert den Bedarf für ein weiterreichendes Konzept.
- **Kapitel 6 (Konzept):** Dieses Kapitel erläutert ein Konzept für die kollaborative Visualisierung großer, interaktiver und dynamischer 3D Szenen auf verteilten Endgeräten, ohne dabei von einer spezifischen Hardware, einem bestimmten Betriebssystem oder einer vorgegeben Programmiersprache auszugehen.
- **Kapitel 7 (Implementierung und Beispiele):** Hier werden die Implementierung des Konzeptes und die dabei festgestellten Erfahrungen und Probleme beschrieben. Zusätzlich befasst sich Kapitel 7 mit Anwendungen, in denen die entwickelte Software bereits erfolgreich eingesetzt wird.
- **Kapitel 8 (Zusammenfassung und Ausblick):** Das abschließende Kapitel gibt eine Zusammenfassung der Ergebnisse der vorliegenden Arbeit sowie einen Ausblick auf zukünftige Aktivitäten.

Kapitel 2

Anforderungen

Die grundlegenden Anforderungen lassen sich direkt aus Kapitel 1.2 ableiten. Die erste Anforderung ist dabei durch die Notwendigkeit bedingt, eine 3D Szene in einem für ein Informationssystem verständlichem Format darzustellen und innerhalb dieses Systems zu verwalten.

Anforderung 2.1 (Repräsentation): Die erste Anforderung ist die Repräsentation von großen, dynamischen und interaktiven 3D Szenen ín einem für das jeweilige Rechnersystem auswertbarem Format.

Da Attribute wie groß, dynamisch und interaktiv sehr dehnbare Bereiche beschreiben können, legt die folgende Aufzählung die Begrifflichkeiten im Rahmen dieser Arbeit fest.

- **Größe:** Große Szenen beinhalten mehrere Millionen Elemente. Elemente können dabei einfache grafische Primitive wie Punkte, Linien, Polygone oder Basiskörper sein, aber auch komplexere Strukturen, welche sich aus mehreren Primitiven zusammensetzen.
- **Dynamik:** Die Dynamik ermöglicht den Elementen einer Szene, ihren Zustand in einer beliebigen Form über die Zeit zu verändern. Darunter fallen nicht nur Transformationen wie Rotation, Skalierung und Translation, sondern gegebenenfalls Farbänderungen oder gar Wertemanipulationen, die für eine der Szene zugrunde liegenden Simulation von Bedeutung sind.
- **Interaktivität:** Die Interaktivität gliedert sich in mehrere Unterbereiche, welche sich aus den an der Interaktion beteiligten Parteien ergeben:
  - **User-Interaktion:** Unter der User-Interaktion ist die freie Navigation, d.h. die Transformation der Betrachterposition und der Blickrichtung, innerhalb der Szene zu verstehen.
  - **User-Element-Interaktion:** Dieser Form der Interaktion liegt das Bedürfnis des Benutzers zugrunde, direkten Einfluss auf das Verhalten eines Elementes zu nehmen oder Informationen abzurufen. Beispiele sind hier das Selektieren und Auswählen von Elementen innerhalb der Szene, um Manipulationen oder Transformationen an den Elementen vorzunehmen, die nicht durch andere Komponenten wie etwa Animationsvorschriften oder Simulationen vorgegeben sind.

  - **Element-Element-Interaktion:** Die Element-Element-Interaktion umfasst alle Interaktionen unter den Elementen einer Szene selbst. Konkrete Anwendungen sind Kollisionserkennungen, aber auch ereignisbasierte Vorgaben, welche durch eine bestimmte Konstellation der Elemente hervorgerufen werden.

Unter dem Punkt der Interaktivität ist auffällig, dass keine User-User-Interaktion aufgeführt ist. Diese ist indirekt gegeben, wenn verschiedene Benutzer innerhalb der gleichen Szene interagieren, die gleichen Elemente manipulieren oder gar durch visuelle Erscheinungsformen wie etwa Avatare repräsentiert sind. Direkte Kommunikationsarten wie der Austausch von E-Mails oder gar das Führen von Video-Konferenzen werden aber nicht weiter berücksichtigt, da sie nicht den Schwerpunkt dieser Arbeit betreffen.

Von der Repräsentation einer Szene bis zu ihrer Visualisierung auf einem Endgerät sind viele Probleme zu bewältigen. Daher ergibt sich die zweite grundlegende Anforderung aus dem Ziel, eine 3D Szene mehreren Benutzern gleichzeitig zugänglich zu machen.

Anforderung 2.2 (Visualisierung): Die zweite Anforderung ist die kollaborative Visualisierung von großen, dynamischen und interaktiven 3D Szenen auf verteilten, heterogenen und möglicherweise mobilen Endgeräten.

Der Begriff kollaborativ umschließt dabei die verschiedenen Formen der Interaktion und garantiert allen Benutzern, das gleiche Maß an Interaktionsmöglichkeiten zu erhalten. Dadurch sind die Benutzer imstande, zusammen innerhalb der Szene zu navigieren und Manipulationen oder Editierungen an den Elementen der Szene vorzunehmen. Analog zu den Attributen der ersten Anforderung erfolgt auch an dieser Stelle eine Definition der Begriffe verteilt, heterogen und mobil.

- **Verteilte Endgeräte:** Das Attribut verteilt impliziert, dass die jeweiligen Endgeräte sich an verschiedenen Lokalitäten befinden können und über ein Netzwerk verbunden sind. Dabei muss es sich nicht zwangsläufig um eine schnelle LAN-Verknüpfung handeln, sondern auch normale Internet- oder Funknetzverbindungen sind inbegriffen.

- **Heterogenität:** Der Begriff der Heterogenität besagt nicht, dass sich die jeweiligen Endgeräte lediglich im Namen des Herstellers unterscheiden, sondern dass die Geräte aus unterschiedlichen Leistungsklassen rekrutieren und entsprechend über variable Leistungsmerkmale verfügen können. Um welche Leistungsmerkmale es sich hier genau handelt, wird noch im weiteren Verlauf dieses Kapitels ausgearbeitet.

- **Mobilität:** Unter mobilen Endgeräten sind im Rahmen dieser Arbeit Laptops und Palmtops zu verstehen. Die Leistungsfähigkeit von mobilen Telephonen ist derzeit noch zu beschränkt, um vertretbare Ergebnisse zu erzielen. Da Laptops aufgrund der limitierten Batteriekapazität und fehlender Funknetzanbindung in der Regel noch stationär verwendet werden, kann darüber diskutiert werden, ob dieser Gerätetyp wirklich zu den mobilen Endgeräten zu zählen ist. An dieser Stelle ist aber nicht die Qualität der mobilen Einsetzbarkeit das Entscheidende, sondern ob und wie sich die zuvor gestellten Anforderungen auf dem jeweiligen Endgerät umsetzen lassen. Zwar ist die Leistungsfähigkeit der Laptops in den letzten Jahren rasant gestiegen, sie steht jedoch stets den

Möglichkeiten eines Personal Computers nach. Aus diesem Grund werden hier Laptops den ebenfalls schwächeren Palmtops zugeschrieben und in der Menge der mobilen Endgeräte vereint.

Wie bereits unter dem Punkt der Heterogenität bemerkt, unterscheiden sich die Leistungsmerkmale der Endgeräte in verschiedenen Aspekten. Daher stellt sich die Frage, welche Merkmale für die Umsetzung der gestellten Anforderungen von Relevanz sind. Anforderung 2.1 definiert unter anderem das Ziel, große 3D Szenen zu repräsentieren. Die Daten einer Szene umfassen dabei nicht nur geometrische Informationen wie Scheitelpunkte, Linien oder Polygone, sondern beispielsweise auch Texturen, Farbtabellen und Materialeigenschaften. Aus diesem Grund können die Informationen mehrere Gigabyte an Speicher beanspruchen, was selbst die Leistungsfähigkeit der meisten heutigen Personal Computer übersteigt. Somit können nur Teilmengen der Daten einer Szene gleichzeitig im Speicher gehalten werden, wohingegen die übrigen Informationen mit Hilfe eines Backup- oder Filesystems verwaltet werden. Je nach Bedarf müssen diese Informationen aus dem Filesystem nachgeladen werden, wodurch ein enormer Zeitaufwand entsteht, da Zugriffe etwa auf eine Festplatte sehr viel langsamer sind als der Zugriff auf eine Speicherzelle. Je nach Größe des Speichers ändert sich auch die Häufigkeit der Dateizugriffe, weshalb die Speichergröße ein zu berücksichtigendes Leistungsmerkmal der Endgeräte darstellt.

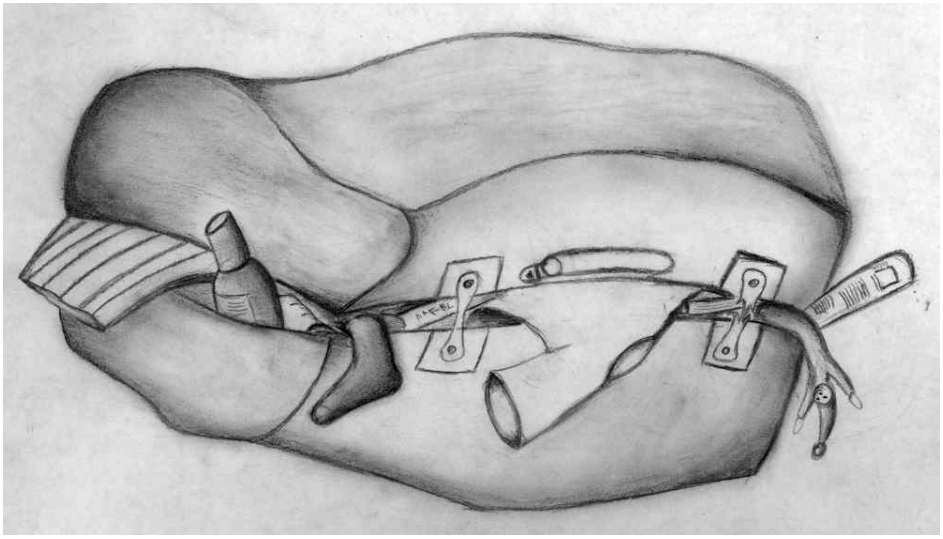

Abbildung 2.1: *Ähnlich diesem Koffer ist auch die Speichergröße derzeitiger Endgeräte limitiert.*

Große Datenmengen implizieren im Allgemeinen auch einen großen Rechenaufwand, beispielsweise bei der Visualisierung einer Szene mit vielen Elementen. Ein wichtige Rolle spielen dabei Floating-Point-Operationen, wie sie bei mathematischen Transformationen auftreten. Andere Aspekte sind die Latenzzeiten der Speicherzugriffe im Rahmen der verwendeten Datenstrukturen zur Verwaltung einer Szene oder der Datendurchsatz des Systembusses, um etwa Speicherblöcke zu verschieben oder in den Speicher der Grafikkarte zu verlagern. In all diesen Fällen können effiziente Algorithmen den Aufwand drastisch reduzieren, letzten Endes sind aber auch sie von der zugrunde liegenden Hardware abhängig. Der Einfachheit halber

werden die verschiedenen Aspekte unter dem Begriff der Rechenleistung eines Endgerätes zusammengefasst und diese als weiteres Leistungsmerkmal eingeführt.

Bei der Analyse innerhalb des Genres der Computerspiele fällt vor allem die rasante Weiterentwicklung der Grafik auf. Möglich wurde dieser Fortschritt durch die Einführung spezieller Grafikhardware, welche den Hauptprozessor bei der aufwendigen Visualisierung der Datenmengen entlastet. Zuvor mussten etwa Routinen zum Rendern von einfarbig gefüllten Polygonen zum einen von Hand programmiert und zum anderen auf Kosten des Hauptprozessors ausgeführt werden. Ein wichtiger Schritt war die Einführung von Standards wie OpenGL (1992) oder Direct3D (1995), welche eine zu implementierende Schnittstelle für die Hersteller von Grafikchips beschreiben. Ein erster Versuch im kommerziellen Bereich war der VESA-Standard, welcher aber von vielen Herstellern nicht unterstützt wurde und sich daher nicht durchsetzen konnte. Grafikbibliotheken in der damaligen Zeit waren oftmals riesige Gebilde mit etlichen Sonderbehandlungen für die jeweiligen Grafikchips. Auch heute noch hängt der Wert eines Grafikchips nicht ausschließlich vom Entwurf des Hardwarelayouts ab, sondern auch von der Qualität der Softwaretreiber, welche die Verbindung zwischen Hardware und Grafikschnittstelle repräsentieren. Einige Endgeräte wie Palmtops bieten immer noch keine spezielle Grafikhardware an, weshalb Visualisierungen auf diesen Geräten entsprechend teuer sind. Als Konsequenz daraus beschränken sich übliche Visualisierungen auf Palmtops auf die Darstellung von Wireframe-Modellen, obgleich Auflösung und Farbtiefe der Displays dieser Geräte mittlerweile durchaus beachtlich sind. Auch die reine Prozessorleistung nimmt erstaunliche Dimensionen an, woran die Bedeutung der Hardware-Unterstützung in Bezug auf die Grafikdarstellung erkennbar ist. Sie ist somit ein weiteres entscheidendes Merkmal.

Personal Computer oder andere stationäre Rechnersysteme bieten meist mehrere Interaktionsmöglichkeiten mit Hilfe verschiedener Eingabegeräte wie Tastatur, Mouse, Joystick oder Tracking Ball an. Dagegen fallen die entsprechenden Optionen eines Palmtops äußerst bescheiden aus, wobei nicht nur der Komfort eines Eingabegerätes entscheidend ist, sondern auch die Erfahrung im Umgang mit selbigem. So ist beispielsweise die Bedienung eines Personal Computer mit grafischer Benutzeroberfläche ohne Mauseingabe äußerst ungewohnt, obwohl der Weg über die Tastatur eigentlich der effizientere sein sollte. Wenn nun mehrere Benutzer mit heterogenen Endgeräten zusammen in einer Szene navigieren wollen, so sind die Interaktionsmöglichkeiten des jeweiligen Endgerätes zu berücksichtigen, da ansonsten die Benutzer mit stationären Systemen bevorteilt wären. Das Ziel kann allerdings nicht sein, diese Benutzer mit einem Handicap zu versehen, um für eine Art Gleichberechtigung zu sorgen. Vielmehr sollte auch für mobile Endgeräte eine effiziente Benutzerschnittstelle angeboten werden. Daher werden die Interaktionsmöglichkeiten eines Endgerätes in den Kreis der Leistungsmerkmale aufgenommen.

Unnabhängig von der Rechenleistung stellt die Qualität der Verbindung zwischen zwei Endgeräten im wahrsten Sinne des Wortes ein wichtiges Bindeglied dar. Das Übertragen großer Datenmengen macht selbst zu einer High-End-Workstation keinen Sinn, wenn der verwendete Kommunikationsweg nur einen geringen Datendurchsatz erlaubt. Dieser Durchsatz wird durch die Bandbreite einer Verbindung beschrieben, dem nächsten Leistungsmerkmal, das allerdings im Gegensatz zu den übrigen nicht gerätespezifisch ist. Wie bereits in Abschnitt 1.2 erklärt, handelt es sich bei der Bandbreite zudem um ein dynamisches Merkmal, was permanenter Überwachung bedarf. Definition 2.1 legt alle eingeführten Leistungsmerkmale fest.

Definition 2.1 (Leistungsmerkmale) Im Rahmen dieser Arbeit sind relevante Leistungsmerkmale die Rechenleistung, der zur Verfügung stehende Speicher, die Hardware-Unterstützung für die Visualisierung, die offerierten Interaktionsmöglichkeiten und die Bandbreite der Netzverbindung.

Es ist offensichtlich, dass die Daten einer Szene sowohl in Quantität als auch in Qualität den Leistungsmerkmalen der Endgeräte und der Netzverbindungen anzupassen sind. Nur so ist eine Kollaboration möglich, da andernfalls die leistungsfähigeren Geräte durch die schwächeren ausgebremst würden. Jeder Benutzer hat aber das Bedürfnis, eine seinem System angemessene Gegenleistung durch die Applikation zu erhalten. Anhand Definition 2.1 lässt sich eine weitere Anforderung ableiten.

Anforderung 2.3 (Adaptivität der Daten) Die Daten einer 3D Szene müssen sowohl in Qualität als auch in Quantität automatisch den Leistungsmerkmalen des jeweiligen Endgerätes angepasst werden.

Eine besondere Bedeutung liegt bei dieser Anforderung in der automatischen Anpassung der Daten. Eine Adaption von Hand würde einerseits einen immensen Aufwand bedeuten und andererseits nur eine spezifische Lösung darstellen, die lediglich für eine bestimme Szene akzeptabel ist. Somit würde es sich nicht um einen generischen Ansatz handeln. Auf einem abstrakten Level lassen sich zwei Möglichkeiten für die Adaption der Datenmenge einer Szene identifizieren, nämlich zum einen die geeignete Selektion von Daten und zum anderen die Reduktion der selektierten Informationen. Erstere gehört zu den nicht-invasiven Techniken für die Beschleunigung des Darstellungsprozesses, d.h. die Verringerung des Aufwands erfolgt ohne eine Manipulation der Daten eines Elements. Ein typisches Verfahren der nicht-invasiven Vorgehensweise ist die Rückseitenentfernung (Backface Culling) bei geschlossenen, nicht durchsichtigen Modellen. Invasive Verfahren jedoch modifizieren die Repräsentation eines Elementes zum Beispiel durch die Entfernung von Geometrie (siehe Kapitel 4.3.1).

Definition 2.2 (Invasive und nicht-invasive Verfahren: Nicht-invasive Verfahren beschleunigen den Darstellungsprozess eines Elementes, ohne dessen Daten zu verändern. Invasive Verfahren dagegen modifizieren die Repräsentation des Elements.

Im Falle der Selektion drängt sich die Frage auf, welche Daten ausgewählt werden müssen beziehungsweise welche Informationen für den jeweiligen Benutzer relevant sind. Ein wichtiges Kriterium ist sicherlich der visuelle Aspekt, d.h. Elemente, die im Sichtbereich des Betrachters liegen, sind mit einer höheren Priorität zu behandeln als Elemente weit außerhalb des Blickfeldes. Es lassen sich aber auch andere Kriterien finden. Beispielsweise können Elemente auch aufgrund einer bestimmten Interaktion mit dem Benutzer von Bedeutung sein. Die Reduktion befasst sich dagegen mit der Detailgenauigkeit der selektierten Elemente. Zum einen müssen vom Betrachter weit entfernte Elemente nicht mit der gleichen Auflösung wie nahe gelegene Elemente visualisiert werden und zum anderen auch nicht mit den gleichen Eigenschaften. So macht beispielsweise das Rendern einer aufwendig texturierten Oberfläche wenig Sinn, wenn von besagter Oberfläche lediglich einige Pixel zu sehen sind. Die Selektion und Reduktion der Information darf nicht nur isoliert auf die Visualisierung einer Szene betrachtet werden, sondern ebenfalls in Hinsicht auf die Übertragung einer Szene. Auch hier gilt die Priorisierung für den Benutzer relevanter Elemente.

Das Beispiel der texturierten Oberfläche leitet zu einer weiteren Anforderung über, welche die Adaption an die Leistungsmerkmale nicht durch die Anpassung der Datenmenge, sondern über die Verwendung angemessener Algorithmen in Angriff nimmt. So macht es beispielsweise Sinn, auf einem Palmtop lediglich Wireframe-Modelle anzuzeigen, obwohl eine Übertragung und Speicherung von Füllfarben durchaus möglich wäre. Wenn jedoch das Rendern des Modells mit gefüllten Polygonen zu einer Framerate von zwei Frames pro Sekunde führt, so ist dies inakzeptabel. Eine entsprechende Anforderung muss dabei nicht zwangsläufig erst auf dem jeweiligen Endgerät ansetzen, sondern eventuell schon bei dem Entscheidungsträger für die Selektion und Reduktion der zu übertragenden Informationen. So ist es beispielsweise überflüssig, Texturen an ein Endgerät zu transferieren, das gar keine Texturen darstellen kann.

Anforderung 2.4 (Adaptivität der Algorithmen) Die Prozessierung der Daten muss durch Algorithmen erfolgen, welche den Leistungsmerkmalen des jeweiligen Endgerätes angepasst sind.

Anhand der bisher aufgestellten Anforderungen wird klar, dass die jeweiligen Endgeräte nicht nur gerenderte 2D Frames erhalten sollen, sondern die wirklichen Elemente der 3D Szene mit ihren Bestandteilen wie etwa Geometrie, Texturen oder Farben. Ansonsten würde es sich in dieser Arbeit um einen Ansatz für das Streamen von Videodaten handeln, einem Gebiet, das bereits durch mehrere Technologien in der Praxis abgedeckt wird. Mit der Software OpenGL VizServer [SGId] bietet SGI eine Client-Server-Architektur, welche mit einem derartigen Ansatz aufwartet. Die Anwender können sich mit ihrem Endgerät zu einem Server verbinden und eine Applikation starten. Der Server berechnet Bild für Bild die Ergebnisse der Anwendung und überträgt diese Bilder an die Endgeräte. Entsprechende Ansätze haben den Vorteil, dass die Berechnung der Applikation nicht auf dem eventuell leistungsarmen Endgerät des Benutzers erfolgen muss. Selbst ein Palmtop ist in der Lage, 2D Bitmaps in einer angemessenen Auflösung zu rendern, weshalb auch hier im Vergleich zu High-End-Geräten ebenbürtige Visualisierungen zu erwarten sind. Jedoch resultieren daraus nicht nur Vorteile. Das Rendern der Bilder auf dem Server erfordert eine immense Rechenleistung, die von einer normalen Workstation nicht mehr geboten wird. Aus diesem Grund offeriert SGI im Paket mit der Software gleich einen Mehrprozessor-Supercomputer, der allerdings neben überragender Performanz auch einen soliden Preis mit sich bringt. Der Origin-3900-Server von SGI beispielsweise kann mit bis zu 128 Mips-Prozessoren und 256 GByte Hauptspeicher ausgerüstet werden, wobei die Sparausgabe mit 64 GByte Hauptspeicher auf schlappe 4,5 Millionen Euro veranschlagt ist. Derartige Summen dürften wohl nur für große Betriebe finanzierbar sein. Der eigentliche Haken aber ist, dass der verwendete Ansatz das grundlegende Problem nicht löst: Die Belastung des Servers wächst mit jedem neuen Client dramatisch. Auch der hier gehandelte Supercomputer wird selbst bei einer schon relativ geringen Anzahl an Clients in die Knie gehen, insofern ein gewisse Qualität der Visualisierung gefordert wird. Das Makabere daran ist, dass die High-End-Workstation eines Benutzers für den gleichen Zweck verwendet wird wie der Laptop oder Palmtop eines anderen Anwenders, nämlich für das Einblenden von 2D-Bitmaps. Was aber spricht dagegen, die Endgeräte der Benutzer in eine Berechnung der Visualisierung im Rahmen ihrer Möglichkeiten einzubeziehen? Ein weiterer Kritikpunkt des behandelten Ansatzes ist die mangelnde Interaktivität von 2D Bildern. Nach den zuvor gestellten Anforderungen würden wirklich die Elemente der 3D Szene übertragen

und wären somit *physikalisch* auf dem Endgerät vorhanden. Daraus ergäben sich wesentliche flexiblere Interaktionsformen mit dem jeweiligen Element, ohne dass eine entsprechende Visualisierung auf dem Server nötig wäre.

Anforderung 2.5 (Erhaltung der Elemente) Die Elemente der 3D Szenen sollen gemäß den Leistungsmerkmalen aus Anforderung 2.1 in ihren Bestandteilen an die Endgeräte übertragen und dort prozessiert werden. Die 3D Szene soll nicht auf einem anderen System bildweise gerendert und dann als Videostream zu dem jeweiligen Endgerät transferiert werden.

Ein weiteres Ziel dieser Arbeit ist außerdem, eine Lösung zu erarbeiten, die auch auf preisgünstigen Personal Computern lauffähig ist. Zudem soll es sich bei dem System, welches die 3D Szenen verwaltet und zur Verfügung stellt, nicht um ein Cluster handeln. Zwar sind die einzelnen Knoten eines Clusters durchaus günstig erhältlich, teuer wird es allerdings bei der Vernetzung der Systeme.

2.1 Zusammenfassung

In diesem Kapitel wurden die grundlegenden Anforderungen an diese Arbeit aufgestellt. Anforderung 2.1 definiert dabei die Attribute der zu repräsentierenden Szenen. Ziel ist die Verwaltung von großen, dynamischen und interaktiven Szenen. Anforderung 2.2 verlangt die kollaborative Visualiserung derartiger Szenen auf verteilten, heterogenen und eventuell mobilen Endgeräten. Zu letzteren werden Laptops und Palmtops gezählt, aber nicht die derzeit noch zu leistungsschwachen mobilen Telefone. Aus dem Begriff der Heterogenität lassen sich verschiedene Leistungsmerkmale der Endgeräte ableiten, die bei der Visualisierung und Übertragung der Szenen zu berücksichtigen sind. Bei diesen Merkmalen handelt es sich um die Rechenleistung, die Speichergröße, die Hardware-Unterstützung in Bezug auf die Grafik, die Interaktionsmöglichkeiten und die Bandbreite als Kriterium für die Qualität einer Netzverbindung. Um eine wirkliche Kollaboration zu erreichen, müssen die Daten der Szenen laut Anforderung 2.3 automatisch an die Leistungsmerkmale der Endgeräte adaptiert werden. Diese Anpassung kann zum einen über die Selektion relevanter Daten und zum anderen über die Reduktion der selektierten Informationen geschehen. Anforderung 2.4 legt eine weitere Vorgehensweise für einen geringeren Rechenaufwand fest, in dem die verwendeten Algorithmen zur Bearbeitung einer Szene ebenfalls den Leistungsmerkmalen der Endgeräte angepasst sein müssen. Aufgrund der eingeschränkten Möglichkeiten von Ansätzen, welche für jeden Client die Bilder einer Applikation auf einem zentralen Server rendern und diese als Videostream übertragen, verlangt Anforderung 2.5 die Übertragung der Elemente einer 3D Szene zu dem jeweiligen Endgerät. Dadurch werden die Endgeräte ihren Möglichkeiten angemessen am Visualisierungsprozess beteiligt. Abschließend wird die Verwendung eines Clusters für die Verwaltung der 3D Szenen ausgeschlossen und das Ziel aufgestellt, eine Umsetzung auf kommerziellen Personal Computern zu erreichen.

Kapitel 3

Anwendungsgebiete und Einsatzmöglichkeiten

Dieses Kapitel geht von einer erfolgreichen Umsetzung der zuvor aufgestellten Anforderungen aus und stellt die Frage, welche Vorteile in diesem Fall entstehen und welcher Nutzen daraus gezogen werden kann. Es steht dabei außer Zweifel, dass die Bedeutung der Grafik und im besonderen die der 3D Grafik während der letzten Jahre in vielen Bereichen erheblich zugenommen hat. Beispiele hierfür sind die Filmindustrie, die Unterhaltungsindustrie oder die Werbebranche. Auch in anderen Gebieten wie etwa im Bereich der Simulationen kommt die grafische Aufbereitung wissenschaftlicher Daten und deren Visualisierung nach einigen Akzeptanzschwierigkeiten allmählich aus den Startlöchern. Gründe für die rasante Verbreitung sind zum einen die gestiegenen Leistungsfähigkeiten der für ein breites Publikum erschwinglichen Hardware und zum anderen die Tatsache, dass der Mensch selbst Teil einer 3D Welt ist und somit über eine entsprechende Wahrnehmung und Erfahrung verfügt. Die folgenden Abschnitte führen nun einige Anwendungsgebiete aus, die nicht auf die bloße Forderung nach einer 3D Visualisierung beschränkt sind, sondern zumindest in mehreren Punkten den Anforderungen aus Kapitel 2 Rechnung tragen. Dabei muss es sich nicht zwangsläufig um vollständig neue Anwendungsgebiete handeln. Es existieren bereits mehrere Bereiche, in denen die Übertragung und Visualisierung von 3D Szenen erfolgreich eingesetzt wird. Allerdings hören diese Umsetzungen auf, sobald es an interessante Punkte wie beispielsweise Größe oder Dynamik einer Szene geht.

3.1 Werbung

Ein Beispiel eines bereits existierenden Anwendungsgebietes ist die Produktwerbung, für deren Präsentation mehrere Firmen Internet-Technologien anbieten. Dabei handelt es sich zumeist um Java-Applets oder kleine Plugins, die heruntergeladen und im Web-Browser des Benutzers installiert werden können. Mit Hilfe dieser Software erhält der Benutzer dann Zugriff auf die Produktinformationen der Webseiten eines Anbieters, sofern dieser die entsprechende Technologie verwendet. Der Benutzer kann in der Palette der Angebote ein Pro-

Abbildung 3.1: *Im Falle der Produktwerbung werden einzelne Elemente zu einem Endgerät übertragen und mit Hilfe eines Browsers angezeigt (Screenshot der Kaon Technologie [KAO]).*

dukt auswählen, worauf eine 3D Repräsentation des Produktes zum System des Benutzers übertragen und angezeigt wird. Sowohl die Qualität der Darstellung als auch die der Interaktivität variiert zwischen den einzelnen Technologien. Beispielsweise glänzt der Hersteller Virtue3D mit Reflexionen auf metallischen Oberflächen, wohingegen andere sich mit einem stufenlosen Zoom der Produkte hervortun. Insgesamt beschränkt sich die Interaktion auf die Vorgabe von einfachen Transformationen wie Rotation oder Skalierung des Artikels, so dass der Benutzer das Produkt von allen Seiten begutachten kann. Einige Technologien erlauben weitergehende User-Element-Interaktionen wie etwa das Öffnen von Autotüren oder das Aufklappen eines Laptops. Die Reaktionen auf diese User-Element-Interaktionen sind gleichbedeutend mit der Dynamik der Szene. Möglichkeiten zur User-Interaktion oder Element-Element-Interaktion werden nicht zur Verfügung gestellt, es sei denn eine Animationshierarchie zwischen den Elementen einer Szene wird als Element-Element-Interaktion erachtet. Die Ursache hierfür ist in der Größe der präsentierten Szenen zu suchen, die sich in der Regel auf ein einzelnes Element oder auf eine kleine Gruppe von Elementen reduzieren. Eine User-Interaktion beziehungsweise eine Navigation innerhalb der Szene ist nicht erforderlich, da mit Hilfe der User-Element-Interaktion bereits die vollständige Begutachtung des Produktes erbracht wird. Element-Element-Interaktionen wie etwa Kollisionserkennungen sind für diese Form der Werbung gleichfalls überflüssig, wenn sämtliche dynamischen Vorgänge durch strikte Animationsvorschriften festgelegt sind.

Durch die Verwendung des Internets als Medium der Übertragung ist die Anforderung nach verteilten Endgeräten gegeben. Allerdings wird nur bedingt Rücksicht auf die Leistungsmerkmale der Endgeräte genommen. Dies beginnt bereits mit der Auswahl der vom Hersteller unterstützen Plattformen. Werden für die Technologie keine Java-Applets sondern Plugins verwendet, so sind diese spezifisch für den jeweiligen Browser. Nur durch ein entsprechend breites Spektrum an Plugins werden die wichtigsten Betriebssysteme berücksichtigt. Anwender von Palmtops gehen dabei sowohl im Falle der Plugins als auch bei der Verwendung von Applets leer aus, weshalb der Aspekt der Mobilität lediglich für Laptops erfüllt ist. Eine

Anforderung	Umsetzung
Größe	Bedingt durch die Präsentation eines einzelnen Artikels sind die Szenen entweder auf ein einzelnes Element oder auf eine kleine Gruppe von Elementen beschränkt, die über eine Animationshierarchie verknüpft sind.
Dynamik	Die Orientierung und Größe der Elemente kann mit Hilfe von einfachen Transformationen verändert werden.
Interaktivität	Über die Transformationen ist lediglich eine User-Element-Interaktion möglich. User-Interaktionen oder Element-Element-Interaktionen treten aufgrund der geringen Anzahl der Elemente nicht auf.
Kollaboration	Es handelt sich um eine Applikation für den Einzelanwender.
Verteilt	Ist gegeben, da die Informationen per Internet an den Anwender übertragen werden.
Heterogenität	Es werden alle Systeme außer Palmtops unterstützt. Allerdings gibt es keine Berücksichtigung der verschiedenen Interaktionsmöglichkeiten eines Endgerätes.
Mobilität	Palmtops werden nicht unterstützt.
Adaptivität der Daten	Die Daten werden nicht den Leistungsmerkmalen der Endgeräte angepasst.
Adaptivität der Algorithmen	Einige Technologien bieten als Option die Visualisierung eines Artikels als texturiertes Modell, als Modell mit gefüllten Flächen oder als Wireframe-Modell.

Tabelle 3.1: *Die Umsetzung der Anforderungen aus Kapitel 2 im Falle der Produktwerbung.*

Adaptivität der Daten existiert nicht. Immerhin bieten einige Hersteller die Option verschiedener Visualisierungsmethoden wie die Darstellung eines Modells als texturiertes Modell, als Modell mit einfarbig gefüllten Flächen oder als Wireframe-Modell. In der einfachen Form der Produktwerbung wird keine Kollaboration unterstützt, was jedoch bei der geringen Zahl an Interaktionsmöglichkeiten verständlich ist. Es existiert allerdings auch komplexere Software, über die beispielsweise das gemeinsame Einrichten eines Raumes mit Mobiliar möglich ist. Tabelle 3.1 gibt einen zusammenfassenden Überblick auf die vorhandene Umsetzung der Anforderungen.

In Hinsicht auf Tabelle 3.1 ist nun die entscheidende Frage, welche Verbesserungen bei einer vollständigen Realisierung der Anforderungen im Bereich der Produktwerbung möglich sind. Eine Option bestünde darin, einfach den Detaillierungsgrad der Elemente zu erhöhen, wodurch sich zumindest von der Quantität weitere Interaktionsmöglichkeiten ergäben. So kann beispielsweise auch ein Auto zu einer großen Szene werden, wenn die einzelnen Komponenten entsprechend zerlegt und repräsentiert werden. Dadurch wären die Hersteller der Technologien zu einer Adaptivität der Daten gezwungen, zumal bereits die derzeitigen einfachen Modelle die Performanz der Rechner strapazieren. Letzten Endes ist aber der Weg der

Abbildung 3.2: *Viele Anbieter unterstützen die Ausführung vordefinierter Animationen wie das Öffnen einer Motorhaube (Screenshot der O2C Technologie [O2C]).*

höheren Detaillierung keine grundlegende Lösung, weil der Aspekt der Kollaboration nicht wesentlich berührt würde. Wie wäre es jedoch mit der Einführung einer geeigneten Testumgebung für das jeweilige Produkt? Diese läge im Interesse sowohl des Produktanbieters als auch des Kunden, kann er sich doch über die Qualität oder Eignung des Kaufobjektes vergewissern. Im Falle des Autoverkaufs könnte der Kunde beispielsweise zu einer virtuellen Testfahrt eingeladen werden. Sicherlich ist die Umsetzung eines authentischen Fahrverhaltens äußerst schwierig, aber der Kunde könnte zumindest einen praxisnäheren Blick auf den Aufbau des Innenraumes, auf die Struktur des Cockpits oder auf den möglichen Sichtbereich werfen. Selbst eine kleine Applikation, welche den Speicherraum des Kofferraum demonstriert, ist denkbar. Mögliche Szenarien für eine virtuelle Testfahrt sind die Simulation innerhalb von Städten oder auf Autobahnen. In beiden Fällen sind sicherlich die Anforderungen nach großen und dynamischen Szenen gegeben, es sei denn der Kunde ist der einzige Autofahrer innerhalb der Stadt. Kollaboration lässt sich über zwei Optionen erreichen. Zum einen könnten sich mehrere Kunden mit verschiedenen Fahrzeugen innerhalb der Szene aufhalten, zum anderen könnte ein potentieller Käufer auch seine Familie zu einer Stadtrundfahrt mitnehmen. Dadurch wäre nicht nur die Position des Fahrers, sondern auch die der Beifahrer abgedeckt. Neben der Kollaboration sind auch Punkte wie die Navigation des Benutzers oder die Element-Element-Interaktion abgedeckt. Schließlich sind auch Kollisionen innerhalb der Szene denkbar. Die Realisierung einer derartigen Testumgebung auf einem Palmtop ist bei den jetzigen Leistungsmerkmalen mit Sicherheit fragwürdig. Andererseits zeigen die vielen hochwertigen Motorsport-Simulationen für den Personal Computer, was in diesem Bereich mittlerweile möglich ist.

Abbildung 3.3: *Die grafische Visualisierung virtueller Chatrooms ist zumeist recht bescheiden (Screenshot aus Active Worlds [ACT]).*

3.2 Virtual Chatrooms

Virtual Chatrooms dienen den Anwendern als virtuelle Treffpunkte im Internet, um Informationen auszutauschen. Die Benutzer können eine Figur als Repräsentation der eigenen Person erstellen, einen sogenannten Avatar, und diese Figur durch eine virtuelle 3D Welt steuern sowie mit den Avataren anderer Anwender Gespräche führen. Ähnlich zu der Produktwerbung bieten die Vertreiber der Chatrooms kleine Software-Plugins an, die als Erweiterung im Web-Browser installiert werden. Mit Hilfe der Plugins kann dann per Browser der Zugriff auf die virtuelle Welt erfolgen. Einige Firmen wie Activeworlds.com Inc. offerieren zudem interaktive Werkzeuge für die Erstellung eigener Welten. Wer eine kreierte Welt der Allgemeinheit zur Verfügung stellen möchte, muss eine Server-Lizenz erwerben, deren Preis sich nach der Größe einer Szene in virtuellen Quadratmetern und der maximalen Anzahl der Benutzer richtet. Insgesamt liegt der Schwerpunkt der virtual Chatrooms in der Kommunikation der Benutzer untereinander und nicht in der Interaktion mit Szenenelementen oder gar in der Element-Element-Interaktion. Die User-Element-Interaktion beschränkt sich oftmals auf das Anklicken von Elementen, die als Links zu weiterführenden Webseiten herhalten. Darüber kann ein Konzern seine Produkte innerhalb der virtuellen Welt feilbieten. Zum Teil sind die Chatrooms nach Berufsgruppen orientiert, so gibt es beispielsweise Welten für Ärzte oder Manager. Die grafische Qualität der Szenen ist äußerst schlicht gehalten, lediglich das Outfit der Avatare sticht mit einem höheren Detailgrad hervor. Eine Adaptivität der Daten und der Algorithmen existiert nicht. Entweder kann das jweilige Endgerät die Daten der Szenen bewältigen oder eine Visualisierung ist nicht möglich, was ein weiterer Grund für die Einfachheit der Szenen ist. Eine Unterstüzung von Palmtops fehlt ebenfalls.

Analog zu Abschnitt 3.1 stellt sich auch hier die Frage, welche Verbesserungen bei einer weitergehenden Umsetzung der Anforderungen aus Tabelle 3.2 möglich sind. Dabei sollte

Anforderung	**Umsetzung**
Größe	Zwar werden von den Dimensionen her betrachtet durchaus weitflächige 3D Welten durch die Szenen repräsentiert, diese sind aber von einer sehr schlichten Detailgenauigkeit, so dass in Bezug auf die Anzahl der Elemente nicht wirklich von großen Szenen gesprochen werden kann.
Dynamik	Die Elemente einer Szene können in der Regel über Skriptsprachen animiert werden.
Interaktivität	Der Schwerpunkt liegt auf der User-User-Interaktion. Zudem ist die User-Interaktion durch die Navigation in der Szene gegeben. User-Element- oder Element-Element-Interaktion werden nicht oder nur bedingt unterstützt.
Kollaboration	Da die User-Element-Interaktion meistens aus weiterführenden Links besteht, ist ein kooperatives Editieren der Szene nicht möglich. Das gemeinsame Navigieren in der Welt ist aber auf alle Fälle vorhanden.
Verteilt	Ist gegeben, da die Informationen per Internet an den Anwender übertragen werden.
Heterogenität	Es werden alle Systeme außer Palmtops unterstützt.
Mobilität	Palmtops werden nicht unterstützt.
Adaptivität der Daten	Die Daten werden nicht den Leistungsmerkmalen der Endgeräte angepasst.
Adaptivität der Algorithmen	Es gibt keine adaptiven Algorithmen.

Tabelle 3.2: *Die Umsetzung der Anforderungen aus Kapitel 2 im Falle der virtual Chatrooms.*

der eigentliche Schwerpunkt der virtual Chatrooms, nämlich die User-User-Kommunikation, erhalten bleiben. Eine Verbesserung der allgemeinen Grafikqualität wäre daher noch keine Lösung, weil sie eventuell von dem eigentlich Sinn und Zweck ablenken könnte. Bei der Analyse einer zwischenmenschlichen Kommunikation fällt auf, dass nicht allein die Sprache entscheidend ist, sondern auch Mimiken, Gestiken, Körperhaltung und Blickkontakt. Von entsprechenden Repräsentationen ist die derzeitige Grafikdarstellung der virtual Chatrooms noch weit entfernt. Ein weiteres Problem ist die mangelnde Personalisierung der Avatare. Entweder muß der Anwender zwischen Standardcharakteren entscheiden oder er kann in einem gesonderten Tool seine eigene Figur designen. Leider sind die Eingabeparameter oftmals so limitiert, dass die Ergebnisse unweigerlich zu muskelbepackten Footballspielern oder zu überdimensionierten Pop-Divas konvergieren. Weiterhin fehlen wichtige Kommunikationsformen wie zum Beispiel das Aufzeichnen oder Malen von schwierigen Sachverhalten, einem Mittel, dessen sich Gesprächspartner häufig bedienen. In all diesen Punkten könnte eine deutlich verbesserte Detaillierung der Szenen weiterhelfen. Grundvoraussetzung dafür ist die Adaptivität der Daten und der Algorithmen, um nicht weiter von der Leistungsvorgabe der schwächsten zu unterstützenden Endgeräte abhängig zu sein. Bei der Adaptivität der Da-

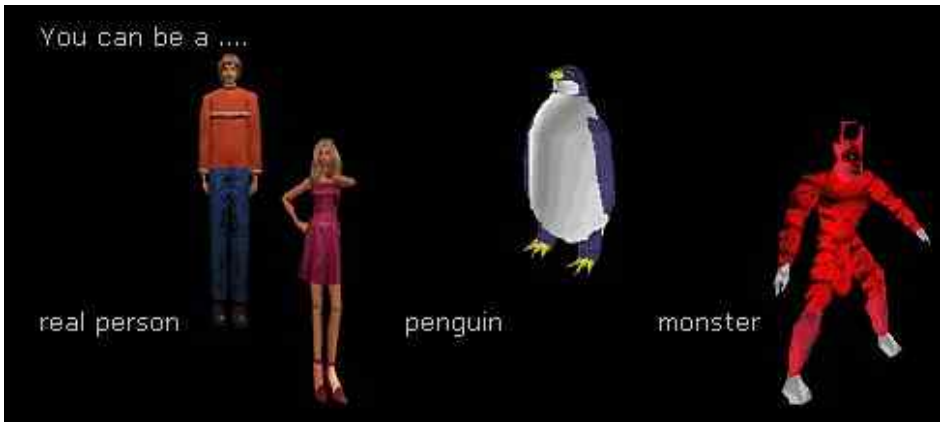

Abbildung 3.4: *Da der Schwerpunkt der virtual Chatrooms in der User-User-Interaktion liegt, ist die Kreation der Avatare oftmals bedeutend vielfältiger als die restliche 3D Welt (Screenshot aus Worlds.com 3D Portal [WOR]).*

ten geht die Selektion der Elemente über räumliche Aspekte hinaus. Für das Darstellen von Gefühls- und Gesichtsausdrücken tragen die Augen- und Mundpartien eine herausragende Funktion und sollten entsprechend priorisiert behandelt werden. Da Mimiken und Gestiken nur schwer intuitiv über Eingabegeräte gesteuert werden können, wäre zudem die Entwicklung eines abstrakten Interaktionsmodells erforderlich.

3.3 Kollaborative 3D Simulation

In weiten Teilen der Industrie spielt die Simulation eine tragende Rolle. Der Begriff *Simulation* wird nach der VDI-Richtlinie 3633 folgendermaßen definiert:

Definition 3.1 (Simulation) Simulation ist das Nachbilden eines Systems mit seinen dynamischen Prozessen in einem experimentierfähigen Modell, um zu Erkenntnissen zu gelangen, die auf die Wirklichkeit übertragbar sind.

Aus Definition 3.1 können bereits einige Punkte herausgedeutet werden, die für die Anforderungen aus Kapitel 2 von Interesse sind. Unter anderem lässt der Begriff "experimentierfähig" einen gewissen Interpretationsfreiraum. Ein Vorteil von computergestützten Simulationen ist, dass ein Vorgang beziehungsweise Versuch mit verschiedenen Parametersätzen beliebig wiederholt werden kann. Für die Vorgabe der Parameter ist eine Editierung der Simulation erforderlich, wodurch wieder der Begriff der Interaktivität in den Vordergrund tritt. Ein anderer wichtiger Aspekt der Definition 3.1 ist die Erwähnung der Dynamik. Somit ist ein weiteres Attribut aus Anforderung 2.1 eingebunden.

Simulationen können sowohl zur Kalkulation geplanter Abläufe im Rahmen einer präventiven Absicherung vor einer teuren Fehlinvestition als auch zur Fehlersuche und Ursachenforschung in bereits existierenden Abläufen dienen. Insbesondere im ersten Fall stellen computergestützte Simulationen eine preisgünstige Lösung dar, da sie nicht nur das Risiko einer Fehlinvestition mindern, sondern zudem ein hohes Maß an Wiederverwendbarkeit offerieren. Abgesehen von den Kosten stellen sie gegenüber der Realität eine sichere Studie dar, die

ohne Gefahr für Mensch und Umwelt durchführbar ist. Ein weiterer Vorteil liegt in der Konzentration auf das Wesentliche, d.h. Simulationen modellieren den Kernpunkt des Interesses.

Bis Mitte der neunziger Jahre wurden die Daten einer Simulation häufig noch als Tabellen oder Diagramme präsentiert. Die Auswertung derartiger Darstellungen ist nicht intuitiv und verlangt die Begutachtung durch Experten. Eine Vermittlung der Informationen an weniger bedarfte Personen gestaltet sich schwierig, was umso schwerwiegender ist, wenn es sich dabei um die Auftraggeber einer Studie handelt. Daher ging man nach einer kurzen Zwischenphase mit 2D Präsentationen zu 3D Visualisierungen über. An dieser Stelle ist es wichtig, zwischen grafischer 3D Simulation und Simulationen mit einer 3D Visualisierung der Ergebnisse zu unterscheiden. Im ersten Fall sind die Simulation, die 3D Szene sowie deren Visualisierung unmittelbar ineinander verzahnt, wohingegen im zweiten Fall die Visualiserung erst nach der Berechnung der Ergebnisse erfolgt. Mittlerweile hat sich die grafische 3D Simulation beziehungsweise Kinematiksimulation als gesonderte Form der Simulation neben Ablaufsimulationen, Finite-Elemente-Simulationen und Mehrkörpersimulationen etabliert. Sie dient in erster Linie der Analyse und Optimierung von Bewegungsabläufen in Produktionssystemen. Alle relevanten Teile der simulierten Anlage werden als 3D-Modelle benötigt, bewegte Teile werden mit ihrer Kinematik beschrieben. Typische Anwendungen sind die Standortoptimierung von Robotern sowie das Erstellen korrekter und kollisionsfreier Steuerungsprogramme für Maschinen und Roboter.

Ein anderes, weit verbreitetes Anwendungsgebiet für 3D Simulationen sind Flugsimulatoren, die in ihrem Spektrum den Anforderungen aus Kapitel 2 am ehesten gerecht werden. Mehrere Benutzer können von ihrem Rechner aus ein Flugobjekt durch eine dynamische, interaktive 3D Welt steuern. Dabei werden insbesondere die Anforderungen 2.1 und 2.2 umgesetzt. Allerdings findet hier keine Übertragung von grafischen Informationen wie Geometrie, Textur oder Topologie statt, sondern diese Daten müssen bereits auf dem jeweiligen Endgerät vorhanden sein. Dieser Punkt leitet zu einem weiteren Problem über, nämlich zu der Adaptivität der Daten und der Algorithmen. Was passiert, wenn ein Endgerät nicht über die erforderliche Kapazität für eine Aufnahme oder Prozessierung der Daten verfügt? Zwar können in der Regel verschiedene Detailstufen der Visualisierung eingestellt werden, bei diesen handelt es sich jedoch nicht um eine automatische Anpassung beliebiger Daten, sondern um vorberechnete Detaillierungsgrade für spezifische Informationen. Im Falle der Visualisierung von statischen geographischen Datensätzen mag dies noch genügen, für die Verarbeitung interaktiver, dynamischer Szenen, in denen beispielsweise auch das Gelände transformiert werden könnte, wäre es jedoch zu inflexibel und aufwändig. Tabelle 3.3 beschreibt die Umsetzung der Anforderungen in Bezug auf die Flugsimulatoren.

Flugsimulatoren sind lediglich ein einzelnes Beispiel für eine umfangreiche Realisierung der Anforderungen. Aufgrund des Vorbildes in der Realität ist es nur eine logische Konsequenz, Aspekte wie Kollaboration oder Interaktivität zu berücksichtigen. Es liegt zudem in der Natur der Sache, große und dynamische Szenen zu repräsentieren. Was aber spricht dagegen, analoges auch für die Kinematikstudie eines Roboters einzusetzen? Experten könnten dann beispielsweise selbst von einem mobilen Kontext heraus die Simulation überprüfen, korrigieren oder editieren. Ein teures Meeting mit entsprechenden Reise- und Aufwandskosten der Experten wäre überflüssig. Gleichzeitig wäre die 3D Visualisierung auch eine intuitive Präsentation für den Laien. Einerseits könnte über verschiedene Visualisierungsformen auf ent-

Anforderung	**Umsetzung**
Größe	Die Repräsentation geographischer Daten für Flugsimulatoren beansprucht durchaus große Datenmengen. Allerdings gibt es lediglich punktuell hohe Auflösungen, beispielsweise im Falle bedeutender Städte. Innerhalb der Städte sind nur wichtige Gebäude detailliert dargestellt. Somit sind die Szenen von der Gesamtzahl der Elemente durchaus groß, verfügen aber nicht über eine hohe Dichte an Informationen.
Dynamik	Bei Simulationen handelt es sich grundsätzlich um die Nachbildung dynamischer Prozesse. Im Fall von Flugsimulatoren werden diese in der Regel durch Fahrzeuge jeglicher Art repräsentiert.
Interaktivität	Es sind alle Formen der Interaktionen aus Kapitel 2 einschließlich der User-User-Interaktion betroffen. Die User-Interaktion entspricht dem Fliegen durch die 3D Szene, die User-Element-Interaktion dem Steuern des Flugobjekts, die Element-Element-Interaktion beispielsweise dem Landen des Objekts auf einer Landebahn und die User-User-Interaktion dem Chatten mit anderen virtuellen Piloten.
Kollaboration	Mehrere Benutzer können gleichzeitig sämtliche Interaktionsformen anwenden.
Verteilt	Die Benutzer sind über Netzwerke verbunden. Allerdings werden nur Kontrollinformationen übersendet und keine grafischen Daten wie Geometrie, Texturen oder Topologien.
Heterogenität	Die Daten können im Rahmen der vorberechneten Detaillierungsgrade der Rechenleistung und der Grafik-Unterstützung der Endgeräte angepasst werden. Palmtops gehen jedoch leer aus.
Mobilität	Palmtops werden nicht unterstützt.
Adaptivität der Daten	Die Daten werden nicht automatisch den Leistungsmerkmalen der Endgeräte angepasst. Stattdessen bleiben in der Regel Details einfach unberücksichtigt.
Adaptivität der Algorithmen	Es gibt durchaus adaptive Algorithmen wie beispielsweise das bilineare oder trilineare Interpolieren von Texturen.

Tabelle 3.3: *Die Umsetzung der Anforderungen aus Kapitel 2 im Falle der Flugsimulatoren.*

scheidende Aspekte hingewiesen werden, beispielsweise über eine entsprechende Kolorierung von Temperaturfeldern. Andererseits könnten aber auch die entscheidenden Informationen selektiert und in einem höheren Detaillierungsgrad dargestellt werden als die nebensächli-

chen Bereiche der Szene. In dem Kontext der mobilen Zugänglichkeit von Informationen gibt es viele denkbare Szenarien. Ein weiteres Beispiel ist die Visualisierung von medizinischen Informationen vor Ort. Ein Arzt oder gar ein Laie könnten 3D Informationen über die Struktur von verletzten Bereichen abrufen. Er könnte die entsprechenden Informationen sogar editieren, um die Art der Verletzung nachzubilden und diese mit Fachkräften kommunizieren. In einem anderen Szenario könnte ein Architekt über eine Baustelle gehen und den 3D Bauplan abrufen, eventuell aufgrund von notwendigen Änderungen editieren und diese mit dem Auftraggeber absprechen. Letzter erhält somit gleichzeitig eine intuitive 3D Präsentation und muss sich nicht allein auf eine sprachliche Beschreibung der Änderungen verlassen.

3.4 Kunst, Design und Unterhaltung

Ein nicht zu unterschätzender Wirtschaftsfaktor liegt im Unterhaltungsbereich und zwar nicht nur in Bezug auf finanzielle Aspekte sondern auch aufgrund wissenschaftlicher Entwicklungen. Insbesonders die Film- und Spielindustrie setzt immer wieder neue Maßstäbe in der kommerziellen Computergrafik. Sie ist sicherlich auch eine bedeutende Motivation für die Hersteller von Grafikchips, innovative Technologien zu erforschen beziehungsweise durch die Hardware zu unterstützen. Filme wie *Titanic* oder *Der Herr der Ringe* bestechen mittlerweile durch derart hochwertige Spezialeffekte, so dass diese kaum noch von der Realität zu unterscheiden sind. Oft werden virtuelle Sequenzen nur noch deshalb als solche erkannt, weil keine rationalen Erklärungen mehr dafür existieren, sei es die Nachbildung von riesigen Steinmonumenten oder die Animation von Figuren aus einer Fantasy-Welt. Die Historie der virtuellen Bilder in der Filmindustrie bringt einige bedeutende Vorteile der computergestützten Grafik zu Tage. Obwohl sich jedes Einzelbild in der Regel von seinem Vor- und Nachfolger nur in geringen Details unterscheidet, wurden früher Cartoons und Comics aus Mangel an Alternativen mit hohem Aufwand von Hand gezeichnet. Heutzutage muss dagegen eine Figur nur einmal als 3D Modell entworfen werden. Für die Animation der Figur können Techniken wie die Skelettanimation oder die inverse Kinematik herhalten. Somit bietet die Computergrafik in diesem Fall eine hohe Wiederverwendbarkeit, da das gleiche 3D Modell für den gesamten Film oder etwaige nachfolgende Serien einsetzbar ist. Die Berechnung von Kameraperspektiven und Lichtquellen sowie das Berücksichtigen von Materialeigenschaften ersetzt das hohe künstlerische Können, was für entsprechende Zeichnungen benötigt wird. Das impliziert aber nicht ein geringeres Niveau für das Design der virtuellen Welten und Figuren. Allerdings können Kameraperspektiven und Lichtquellen mit Hilfe einfacher Parametrisierungen für die Produktion ganzer Sequenzen verwendet werden. Der Einsatz mehrerer Kameras erlaubt zudem das Einblenden einer Szene aus verschiedenen Perspektiven unter Verwendung der gleichen Szenendaten. Die Computergrafik ist daher eine wesentlich flexiblere Lösung als das Zeichnen von Einzelbildern. Sie bietet außerdem eine Qualität, welche für den Menschen in vergleichbarer Zeit nicht realisierbar wäre. Daraus lässt sich auch ein effektiverer Kosten-Nutzen-Wert ableiten. Dem stehen jedoch die gestiegenen Anforderungen und die damit verbundenen Datenmengen gegenüber. Für die Berechnung von Filmsequenzen werden heutzutage aufwendige und teure Rechnerfarmen benötigt, die

trotz ihrer gigantischen Rechenleistung mehrere Monate für die Produktion der Einzelbilder aufwenden müssen.

Abbildung 3.5: *3D Online Rollenspiele wie Ragnarok verfügen mittlerweile über eine gute Grafikqualität, wobei jedoch hier auch keine Übertragung der visuellen Informationen einer Szene erfolgt.*

Filme wie *Der Herr der Ringe* verlangen sogar noch weitergehende Technologien. Für die aufwändigen Kampfszenen war es unmöglich, mehrere tausende von Statisten und Schauspielern mit der notwendigen Erfahrung und Disziplin aufzubringen. Die Konsequenz daraufhin war, die Figuren durch virtuelle Charaktere zu ersetzen. Sowohl das Modellieren der Figuren als auch deren Verhalten hätte aber für jeden einzelnen Charakter einen riesigen Aufwand bedeutet. Andererseits durften die Modelle jedoch auch nicht zu oft reproduziert werden, um nicht den Eindruck der Gleichförmigkeit zu erwecken. Aus diesem Grund wurden mehrere Prototypen mit unterschiedlichem Aussehen geschaffen und mit Hilfe der Agententechnologie repräsentiert. Jeder Agent entsprach einem Charakter und erhielt obendrein einen Satz möglicher Aktionen, die er aufgrund des Eintretens bestimmter Ereignisse ausführen konnte. Dazu gehörten Aktionen wie das Angreifen von Gegnern, das Flüchten vor Gegnern oder das Erklettern von Leitern bei einer Belagerung. Für die Filmsequenz wurden die Charaktere beider Parteien in der Szene platziert und dann die Simulation gestartet. Auch wenn die Grafik der Simulation sicherlich nicht den endgültigen Bildern entsprach, handelt es sich dennoch um eine Anwendung großer, dynamischer und interaktiver Szenen. In einem derartigen Einsatzzweck wäre die Forderung nach Mobilität wohl etwas übertrieben, da es sich hier um langfristig geplante Vorgänge handelt. Außer der Element-Element-Interaktion zwischen den

Charakteren der Szene wäre aber eine User-Element-Interaktion verbunden mit einer Kollaboration durchaus wünschenswert. Beispielsweise könnten darüber verschiedene Anwender von ihrem Endgerät aus bestimmte Charaktere gezielt steuern und so den Handlungsablauf der Sequenz entscheidend beeinflussen.

Ähnliche Szenarien finden sich ebenfalls in 3D Online-Spielen, die sich insbesondere im Bereich der Rollenspiele einer wachsenden Fan-Gemeinde erfreuen. Die Gründe hierfür liegen in der wesentlich leistungsfähigeren und kostengünstigeren Verfügbarkeit des Internets. Noch vor einigen Jahren zeichneten sich Online-Spiele vor allem durch eine äußerst bescheidene Grafik aus, die dem Standard herkömmlicher Software beträchtlich nachstand. Durch den Support von Seiten der Grafikchips änderte sich das zunehmend, so dass Spiele wie Worlds of Warcraft, Everquest II oder The Saga of Ryzom in ihrem Metier neue Referenzen setzen werden. Analog zu den Flugsimulatoren müssen auch hier Kompromisse getroffen werden. Die Nachbildung großer, dynamischer und interaktiver Welten beansprucht die Rechenleistung derzeitiger Personal Computers in extremer Art und Weise, weshalb in bestimmten Punkten Abstriche in Kauf genommen werden müssen. Beispiele hierfür sind die Konzentration auf wichtige Anlaufpunkte wie Städte oder Gebäude, die in einem ungleich höheren Detaillierungsgrad dargestellt werden als die Gebiete zwischen den Ballungszentren. Andere Kompromisse betreffen die Interaktions- und Verhaltensmöglichkeiten der Charaktere, die in vielen Spielen ihr gesamtes Dasein anscheinend Tag und Nacht am gleichen Ort fristen. Allerdings wäre die komplexe Interaktion zwischen tausenden von Charakteren bei gleichzeitiger Hochglanzgrafik für derzeitige Systeme nicht durchführbar. Übereinstimmend mit den in diesem Kapitel vorgestellten Anwendungsgebieten fehlt auch hier eine automatische und generische Adaptivität der Daten, so dass lediglich vorberechnete Detailstufen verwendet werden beziehungsweise unberücksichtigt bleiben können. Die Adaptivität der Algorithmen beruht neben der bereits erwähnten Filterung der Texturen ebenfalls auf dem Wegfall aufwändiger Berechnungen wie etwa Schattenwürfen. Zudem fehlt eine Übertragung grafischer Daten, weshalb die entsprechenden Daten bereits auf dem jeweiligen Endgerät existieren müssen. Das geschieht auch heute noch hauptsächlich über den Verkauf von CDs. Etwaige Erweiterungen der Szenen erfolgen über den Erwerb sogenannter Zusatz-CDs. Somit ist die Flexibilität in der Manipulation der Welten beschränkt. Tabelle 3.4 wirft einen zusammenfassenden Überblick auf die Umsetzung der Anforderungen aus Kapitel 2 in Bezug auf 3D Online-Spiele.

Ein weiteres Anwendungsgebiet liegt im kollaborativen Design von Produkten, beispielsweise in der Automobilbranche. Hier werden hochaufgelöste Nachbildungen der Karossen generiert und aerodynamischen Tests unterworfen. Bei einer Umsetzung der Anforderungen könnte der Entwurfsprozess weiter vereinfacht werden. Verschiedene Designer könnten dann ein Produkt von ihrem jeweiligen Endgerät aus bearbeiten, ohne dabei vor Ort zu sein. Entscheidungen wären sogar in einem mobilen Kontext möglich, da eine dem Endgerät entsprechende 3D Visualisierung zur Verfügung stünde.

Anforderung	**Umsetzung**
Größe	Ähnlich den Flugsimulatoren werden auch hier sehr weitläufige Arreale repräsentiert. Allerdings mangelt es mit der Ausnahme von Ballungszentren wie Städten oder Ortschaften ebenfalls an einer hohen Dichte der Informationen.
Dynamik	Zumindest die Avatare der Benutzer innerhalb der Szene sind dynamisch. Je nach Produkt können aber auch Gegenstände benutzt und verändert werden, beispielsweise das Öffnen von Türen, das Aufnehmen oder Ablegen von Waffen und eventuell sogar das Backen von Brot.
Interaktivität	Alle Interaktionsformen einschließlich der User-User-Interaktion werden unterstützt. Der Benutzer steuert einen Avatar durch eine 3D Welt (User-Interaktion), nimmt Gegenstände auf (User-Element-Interaktion) oder lässt sie auf den Boden fallen (Element-Element-Interaktion und unterhält sich mit den Avataren anderer Benutzer (User-User-Interaktion).
Kollaboration	Mehrere Benutzer können gleichzeitig sämtliche Interaktionsformen anwenden. Dadurch können sie gemeinsam die 3D Welt im Rahmen der angebotenen Möglichkeiten editieren.
Verteilt	Zwar sind die Benutzer über Netzwerke verbunden, jedoch werden keine grafischen Informationen übertragen sondern nur Kontroll- oder Transformationsanweisungen. Die 3D Welt muss daher bereits vor Beginn des Spiels auf dem jeweiligen Endgerät zur Verfügung stehen.
Heterogenität	Die Daten können im Rahmen der vorberechneten Detaillierungsgrade der Rechenleistung und der Grafik-Unterstützung der Endgeräte angepasst werden.
Mobilität	Palmtops werden nicht unterstützt.
Adaptivität der Daten	Die Daten werden nicht automatisch den Leistungsmerkmalen der Endgeräte angepasst. Stattdessen bleiben in der Regel einfach Details unberücksichtigt.
Adaptivität der Algorithmen	Es gibt teilweise adaptive Algorithmen wie beispielsweise das bilineare oder trilineare Interpolieren von Texturen. Hauptsächlich werden aber aufwendige Berechnungen wie etwa Schattenwürfe einfach ausgeschaltet.

Tabelle 3.4: *Die Umsetzung der Anforderungen aus Kapitel 2 im Falle der 3D Online-Spiele.*

3.5 Zusammenfassung

In diesem Kapitel wurden verschiedene Anwendungsgebiete im Falle einer erfolgreichen Umsetzung der Anforderungen aus Kapitel 2 vorgestellt. Die Beispiele wie etwa die Produktwerbung, die virtuellen Chatrooms oder die 3D Simulationen machen deutlich, dass in dieser Thematik bereits Ansätze existieren und ein großer Bedarf an entsprechenden Techniken besteht. Allerdings werden bei der Lösung der Problematiken fast immer stereotypische Ansätze verfolgt, weshalb sich die Ergebnisse häufig gleichen. So bietet etwa keine Technologie eine automatische Adaptivität der Daten, die über im Voraus entworfene Detaillierungsgrade hinausgeht. Weiterhin sind in fast allen Fällen die Möglichkeiten zur Interaktion oder Kollaboration eingeschränkt. Andere häufig vernachlässigte Aspekte betreffen die Übertragung von geometrischen Informationen, deren Anpassung an die gegebene Bandbreite und die Integration mobiler Endgeräte.

Kapitel 4

Grundlagen

Neben den möglichen Anwendungsgebieten und Einsatzmöglichkeiten stellt sich die Frage, welche Technologien für eine erfolgreiche Umsetzung der Anforderungen aus Kapitel 2 erforderlich sind. Diese Technologien sollen in dem folgenden Kapitel herausgearbeitet und sofern für das weitere Verständnis der Arbeit relevant vorgestellt werden.

4.1 Elementrepräsentation

In den vorangehenden Abschnitten wurde bereits mehrmals der Begriff der Elemente einer Szene verwendet, bei denen es sich laut Kapitel 2 einerseits um einfache grafische Primitive wie Punkte, Linien, Polygone oder Basiskörper handeln kann, aber andererseits auch um komplexere Strukturen, welche sich aus mehreren Primitiven zusammensetzen. Tabelle 4.1 klassifiziert wichtige Ansätze zur Modellierung und Repräsentation der Elemente einer 3D Szene in vier grundlegende Teilbereiche. Die Gruppe der Rohdaten umfasst Ansätze mit unstrukturierten Datenmengen wie etwa Punktwolken oder Polygonmengen. Die zweite Gruppe versucht, die Elemente einer Szene mit flächenbasierten Techniken zu modellieren und gehört wohl zu der am häufigsten frequentierten Vorgehensweise. Eine weitere Art von Ansätzen repräsentiert Elemente mit Hilfe von Volumenbeschreibungen. Unter den aufgeführten Techniken dieser Gruppe ist für diese Arbeit vor allem der Binary Space Partioning Tree (BSP) hervorzuheben, allerdings weniger aufgrund seiner Modellierungseigenschaften. Vielmehr können Space Partitioning Trees sehr vielseitig eingesetzt werden, unter anderem wie in Abschnitt 4.4 beschrieben auch für eine räumliche Sortierung der Elemente einer Szene im Sinne der vierten Gruppe. Diese beinhaltet Vorgehensweisen zur Repräsentation komplexer Strukturen beziehungsweise zur Beschreibung von Szenen mit mehreren oder vielen Elementen.

Da die Forschungsarbeiten und -ergebnisse innerhalb der einzelnen Teilbereiche sehr umfangreich und vielfältig sind, sollen an dieser Stelle nur Techniken weiter ausgeführt werden, welche für den Rahmen dieser Arbeit von Bedeutung sind. Dabei handelt es sich insbesondere in der zweiten Gruppe um die Polygonnetze, in der dritten Gruppe um die Space

Rohdaten	**Flächen**
Punktwolke	Polygonnetze
Range image	Unterteilungsflächen
Polygon Mengen	Parametrische Flächen
	Implizite Flächen
Volumenmodelle	**High-Level Strukturen**
Voxel	Szenegraph
Binary Space Partitioning Tree	Skelette
Constructive Solid Geometry	Anwendungsspezifische Strukturen
Sweeps	

Tabelle 4.1: *Die Elemente einer Szene können mit Hilfe mehrerer Ansätze repräsentiert werden.*

Partitioning Trees und in der vierten Gruppe um die Szenegraphen. Aufgrund ihrer Bedeutung für eine automatische Adaption geometrischer Daten werden außerdem die übrigen Ansätze der zweiten Gruppe kurz vorgestellt. Grundsätzliches Ziel dieser Arbeit ist jedoch die Unabhängigkeit von spezifischen Technologien für die Repräsentation der Elemente, d.h. auch der Einsatz aller anderen Techniken wie etwa der Constructive Solid Geometry soll ohne Auswirkung auf die Gesamtarchitektur möglich sein.

4.1.1 Polygonnetze

Polygonenetze beziehungsweise Meshes sind auch heute noch trotz der zunehmenden Verwendung von kurvenkontrollierten Flächen und Objekten wie etwa den Non-Uniform Rational B-Splines (NURBS) in vielen populären CAD-Systemen weit verbreitet. Sie bieten eine einfache und intuitive Repräsentation zur Beschreibung von Oberflächen mit vielfältigen Möglichkeiten zur Manipulation. Einer der Hauptgründe für den Einsatz von Polygonnetzen liegt aber in ihrer direkten Unterstützung durch die Grafik-Hardware, deren Visualisierungsmechanismen sich fast ausschließlich auf die Ausgabe von Dreiecken beschränken. Dreiecke bieten gegenüber anderen n-Ecken die vorteilhaften Eigenschaften, stets sowohl planar als auch konvex zu sein. Entsprechend finden Dreiecksnetze als spezialisierte Form der Polygonnetze die häufigste Verwendung. Da eine Generierung von Hand oftmals zu viel Aufwand bedeutet, werden sie heutzutage unter anderem automatisch durch den Einsatz von 3D Scannern mit sehr hohen Auflösungen erzeugt. Die Notwendigkeit derartiger Auflösungen bringen das Problem der Polygonnetze zu Tage, gekrümmte Flächen nicht exakt darstellen, sondern nur annähern zu können. Entsprechend verschlingen gute Approximationen eine gewaltige Speichermenge, was den Einsatz der Polygonnetze für die Repräsentation und Übertragung geometrischer Informationen nicht gerade leichter macht. Zudem sind sie lediglich zur Modellierung endlicher Bereiche geeignet.

Da Polygonnetze einen gerichteten Graphen beschreiben, können die aus der Graphentheorie bekannten Definitionen übernommen werden.

Definition (Graph): Ein Graph $G = (V, E)$ besteht aus einer nichtleeren Knotenmenge

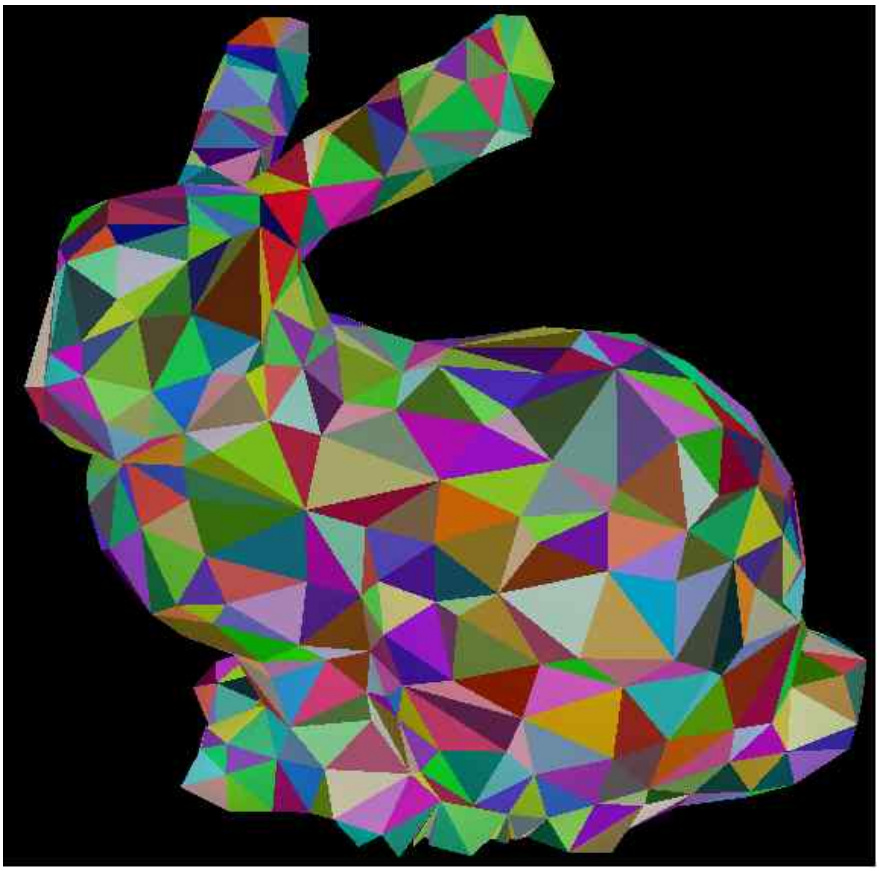

Abbildung 4.1: *Der bekannte Stanford Bunny ist durch ein Netz aus Dreiecken beschrieben.*

V, einer Kantenmenge E sowie einer auf E definierten Abbildung ω (Inzidenzabbildung), die jeder Kante aus E genau ein Knotenpaar $\{v_i, v_j\}$ mit $v_i, v_j \in V$ zuordnet. In einem gerichteten Graphen $G = (V, E)$ ist das jeder Kante aus E zugewiesene Knotenpaar $\{v_i, v_j\}$ geordnet.

Die Punkte eines Polygonnetzes sind innerhalb des Graphen durch jeweils einen Knoten repräsentiert, der auch als Scheitelpunkt (Vertex) bezeichnet wird. Jedes Polygon innerhalb des Netzes wird über einen geschlossenen Weg beziehungsweise Zyklus des Graphen beschrieben.

Definition (Zyklus): Sei $G = (V, E)$ ein gerichteter Graph. Ein Kantenzug $v_1, \ldots, v_k$ heißt ein Weg von v_1 nach v_k in G, wenn die Kanten paarweise verschieden sind. Der Kantenzug $v_1, \ldots, v_k$ heißt gerichteter Weg von v_1 nach v_k in G, wenn $\{v_i, v_{i+1}\} \in E$ für alle $i = 1, \ldots, k-1$ und die Kanten paarweise verschieden sind. Die Länge des Weges ist $k-1$. Ein gerichteter Weg mit $v_1, \ldots, v_k$ mit $k \geq 2$ heißt Zyklus in G, wenn $v_k = v_1$ gilt.

Der Bezug zwischen Zyklus und Polygon gilt jedoch nur für $K \geq 3$, da bereits für Dreiecke mehr als zwei Kanten benötigt werden. Die Definition der Polygone sollte in einer einheitlichen Orientierung erfolgen, um etwa die konsistente Berechnung der Normalen zu ermöglichen. Ansonsten kann es unter anderem bei Licht- und Reflexionsberechnungen zu fehlerhaften Darstellungen kommen. Zumindest erfordern viele Algorithmen die eindeutige Orientierbarkeit aller Polygone eines Netzes, wie es beispielsweise im Falle des Möbiusbandes

oder der Klein'schen Flasche nicht gegeben ist.

Das durch die Modellierung eines Elementes entstehende Polygonnetz muss aufgrund der Vorgabe des Originals nicht unbedingt zusammenhängend sein. Wird beispielsweise ein Schlagzeug als eine geometrische Einheit verstanden, die durch ein einzelnes Mesh repräsentiert werden soll, dann besteht der resultierende Graph aus mehreren Komponenten, die jeweils die physikalisch getrennten Instrumente des Schlagzeugs beschreiben. Im Falle dieser Komponenten des Graphen wird auch von Zusammenhangskomponenten gesprochen. Die Zusammenhangskomponenten eines Graphen verfügen über disjunkte Knotenmengen, wobei keine Kante von einem Knoten einer Zusammenhangskomponente S_1 zu einem Knoten einer Zusammenhangskomponente S_2 existiert.

Definition (Zusammenhangskomponente): Sei $G = (V, E)$ ein gerichteter Graph. G heißt zusammenhängend, wenn es zu je zwei Knoten $v, w \in V$ einen Kantenzug von v nach w gibt. Eine Zusammenhangskomponente ist eine maximale Teilmenge unter den Teilmengen der Knotenmenge mit der Eigenschaft, dass der induzierte Teilgraph $G' = (V', E')$ mit $V' \subseteq V$ und $E' \subseteq E$ zusammenhängend ist.

Bei der Modellierung einer Papierseite und einer Sphäre fällt ein weiteres wichtiges Merkmal der Polygonnetze auf. Während im Falle des Papiers die Beschreibung mit Hilfe eines offenen Meshes möglich ist, bietet sich für die Repräsentation der Kugel ein geschlossenes Mesh an. Im Gegensatz zu geschlossenen Polygonnetzen verfügen offene Meshes über Ränder, d.h. Kanten, die nicht adjazent zu mehreren Polygonen sind. Obwohl ein Torus ebenfalls über ein geschlossenes Mesh dargestellt werden kann, unterscheidet er sich dennoch von der Kugel in mehr als dem optischen Eindruck. Worin bestehen nun diese Unterschiede?

Da ein Polygonnetz eine gekrümmte Fläche nur approximieren kann, muss für die Modellierung der Sphäre eine geeignete Darstellung gefunden werden. Hierzu bieten sich konvexe Polyeder an, die wie folgt definiert sind.

Definition (Polyeder): Ein Polyeder ist eine beschränkte, abgeschlossene Teilmenge des R^3. Seine Oberfläche ist eine geschlossene, orientierbare 2-Mannigfaltigkeit, d.h. jeder Punkt auf der Oberfläche besitzt eine Umgebung (Nachbarschaft), die stetig verformbar (homöomorph) zu einer offenen Scheibe des R^2 ist. Ein Polyeder ist aus endlich vielen, planaren Polygonen zusammengesetzt, die jeweils von endlich vielen Kanten begrenzt sind.

Für die Definition eines konvexen Polyeders fehlt noch die Definition der Konvexität eines d-dimensionalen Elements.

Definition (Konvexität): Ein d-dimensionales geometrisches Element ist konvex, wenn die Strecke zwischen je zwei Punkten seiner $(d-1)$-dimensionalen Oberfläche selber vollständig innerhalb des Elements liegt.

Alle konvexen Polyeder sind homöomorph zu einer Kugel (kugelähnlich), wobei der Kantengraph aller kugelähnlichen Polyeder planar ist.

Definition (Planarer Graph): Ein Graph heißt planar, wenn er ohne Überschneidungen seiner Kanten in die Ebene eingebettet werden kann.

Im Gegensatz zu den Graphen konvexer Polyeder kann der Kantengraph des obendrein nicht konvexen Torus nur mit Überschneidungen in eine Ebene eingebettet werden. Dieser Unterschied zwischen Kugel und Torus ist über das Geschlecht (Genus) eines Graphen definiert.

Definition (Geschlecht eines Graphen): Sei M_k die kanonische geschlossene, orientierbare 2-Mannigfaltigkeit vom Geschlecht k. M_k entspricht dabei einer Kugel mit k Löchern beziehunsweise einer Kugel mit k Henkeln. Ein Graph ist vom Geschlecht k, wenn er überschneidungsfrei in M_k eingebettet werden kann, aber nicht in M_{k-1}.

Die vom Torus umschlossene Öffnung wird als (topologisches) Loch bezeichnet, weshalb der Graph eines Torus vom Geschlecht $H = 1$ ist. Eine Kugel dagegen verfügt über kein derartiges Loch. Sie ist somit vom Geschlecht $H = 0$.

Nach dem Invarianztheorem in der Charakteristik von Euler kann für einfach zusammenhängende Flächen (Polyhedrons) die folgende Beziehung zwischen der Menge V der Scheitelpunkte, der Menge E der Kanten und der Menge P der Polygone aufgestellt werden.

$$|V| + |P| - |E| = 2 \tag{4.1}$$

Da dies für viele Polygonnetze nicht ausreichend ist, erlaubt die erweiterte *Euler-Poincaré-Formel* eine Abschätzung auch für nicht einfach zusammenhängende beziehungsweise mehrfach zusammenhängende Flächen, wobei S der Anzahl der Zusammenhangskomponenten, H dem Genus und B der Menge der Ränder eines Meshes entspricht.

$$|V| + |P| - |E| = 2(S - H) + |B| \tag{4.2}$$

Für einen zusammenhängenden Graphen kann $S = 1$ eingesetzt werden.

$$|V| + |P| - |E| = 2 - 2H + |B| \tag{4.3}$$

In einem geschlossenen 2-mannifold Mesh kann die Anzahl der Kanten in Relation zu der Anzahl der Polygone und der durchschnittlichen Anzahl der Scheitelpunkte pro Polygon $|V_p|$ gesetzt werden. Bei der separaten Aufsummierung der Kanten für jedes Polygon wird jede Kante innerhalb des Meshes zweimal berücksichtigt.

$$|E| = |P|\,|V_p|\,/2 \tag{4.4}$$

Analog kann die Anzahl der Kanten über die Anzahl der Scheitelpunkte und der durchschnittlichen Anzahl der adjazenten Polygone pro Scheitelpunkte beschrieben werden.

$$|E| = |V|\,|P_v|\,/2 \tag{4.5}$$

Wird $|E|$ aus Gleichung 4.3 über Gleichung 4.5 substituiert, so ergibt sich

$$\begin{aligned} |V| + |P| - |V|\,|P_v|\,/2 &= 2 - 2G + |B| \\ |V| + |V|\,|P_v|\,/\,|V_p| - |V|\,|P_v|\,/2 &= 2 - 2G + |B| \\ 1/\,|P_v| + 1/\,|V_p| + 1/2 &= (2 - 2G + |B|)/(|V|\,|P_v|) \end{aligned}$$

Da der rechte Term in der letzten Gleichung bei einer großen Menge V gegen 0 konvergiert, kann die folgende Gleichung als Annäherung für große Meshes verwendet werden.

$$1/\,|P_v| + 1/\,|V_p| = 1/2 \tag{4.6}$$

In einem Dreicksnetz wird somit ein Scheitelpunkt im Durchschnitt von 6 Dreiecken referenziert, in einem Viereckснetz immerhin noch von 4 Polygonen. Diese Werte entsprechen dem durchschnittlichen Grad der Scheitelpunkte in dem jeweiligen Netz.

Definition (Grad eines Knoten): Sei $G = (V, E)$ und v ein Knoten in G. Der Eingangsgrad von v ist die Anzahl der Kanten $\{w, v\} \in E$. Der Ausgangsgrad von v ist die Anzahl der Kanten $\{v, w\} \in E$. Der Grad von v ist die Summe aus Eingangsgrad und Ausgangsgrad von v.

Werden die Grade für $|P_v|$ in Gleichung 4.6 eingetragen, so nähert sich die Anzahl der Kanten in einem Dreiecksnetz $3|V|$ und in einem Viereckснetz $4|V|$ an.

Prinzipiell ist es möglich, jedes Polygon über die Koordinaten seiner Scheitelpunkte zu beschreiben und analog im Speicher zu verwalten. Da sich in Polygonnetzen aber die Polygone Scheitelpunkte mit anderen Polygonen teilen, würde dieses Verfahren zu einem unnötigen Speicherverbrauch führen. Ein weiterer Grund für eine alternative Vorgehensweise ist die Notwendigkeit von Zusatzinformationen wie etwa Nachbarschaftsbeziehungen, welche insbesondere bei der Vereinfachung von Polygonnetzen eine wichtige Rolle spielen. Die Lösung zu beiden Problemen liegt in der indizierten Speicherung der Scheitelpunkte, welche auch als Shared Vertex Speicherung bekannt ist. Anstatt für jedes Dreieck die Koordinaten seiner Scheitelpunkte aufzulisten, werden alle Vertices jeweils einmal in einer gesonderten Vertexliste gespeichert. Diese Vertexeinträge werden durch die Dreiecke über Indizes referenziert, welche normalerweise eines deutlich geringeren Speicheraufwandes bedürfen als ein vollständiger Scheitelpunkt. Beinhaltet ein Scheitelpunkteintrag außer den Koordinaten Indizes auf seine adjazenten Polygone, so ist darüber ein Bezug sowohl auf die Anzahl dieser Polygone als auch auf die Polygone selbst gegeben.

Übliche Vorgehensweisen bei der Verwaltung von Polygonnetzen können in punkt-, kantenbeziehungsweise polygonbasierte Ansätze klassifiziert werden. Im folgenden werden einige dieser Strukturen vorgestellt und unter anderem in Bezug auf ihre Speichereffizienz analysiert. Letztere ist jedoch im Zusammenhang mit der Zugänglichkeit topologischer Informationen innerhalb eines Meshes zu sehen. Wenn eine bestimmte Information über eine möglichst geringe Anzahl von Zugriffen abrufbar sein soll, dann ist eventuell ein erhöhter Speicherverbrauch in Kauf zu nehmen.

4.1.2 Kantenbasierte Strukturen

Eine weit verbreitete Repräsentation von 2-mannifold Meshes ist die kantenbasierte Winged Edge Struktur [Bau75]. Sie enthält eine Liste für die Polygone, eine Liste für die Kanten und eine Liste für die Scheitelpunkte. Jeder Polygoneintrag referenziert eine Kante des korrespondierenden Polygons und jeder Scheitelpunkteintrag eine adjazente Kante. Die Kanten selbst beinhalten vier Referenzen auf die vorangehenden beziehungsweise nachfolgenden Kanten innerhalb der beiden adjazenten Polygone, zwei Referenzen auf die Scheitelpunkte und nochmals zwei Referenzen auf die Polygone selbst. Somit verfügt diese Struktur zwar über umfassende Informationen der Topologie eines Netzes, sie ist aber auch entsprechend speicheraufwändig. Beispielsweise wären in einem großen Viereckesnetz ungefähr $16|V|$ Referenzen erforderlich, in einem Dreiecksnetz gar etwa $24|V|$.

4.1.3 Polygonbasierte Strukturen

In polygonbasierten Datenstrukturen werden in jedem Polygoneintrag Referenzen auf die zugehörigen Kanten und Scheitelpunkte verwaltet. Jeder Scheitelpunkteintrag beinhaltet eine Referenz auf eine korrespondierende Kante und eine Referenz auf das adjazente Polygon. Somit werden pro Polygoneintrag drei Referenzen für je einen Scheitelpunkt benötigt, was ausgehend von den Gleichungen 4.4 und 4.5 zu folgender Abschätzung führt.

$$|P|\,(3\,|P_v|) = 3\,|V|\,|P_v| \tag{4.7}$$

Da in großen Dreiecksnetzen $|P_v|$ durch 6 und in Viereckesnetzen durch 4 ersetzt werden kann, ergeben sich für erstere $18|V|$ und für letztere $12|V|$ Referenzen. Zwar ist dies bereits effizienter als bei der Winged Edge Struktur, allerdings kann der Aufwand noch weiter reduziert werden.

Polygonstreifen speichern benachbarte Polygone mit mindestens einer gemeinsamen Kante. Durch die Sortierung der Scheitelpunkte ergibt sich eine Kante des aktuellen Polygons implizit aus einer Kante des Vorgängers. Da mit jeder Kante zwei Scheitelpunkte eingespart werden können, reduzieren sich die Speicherkosten bei Dreiecksnetzen im Vergleich zu Winged Edge Strukturen auf zwei Drittel und bei Viereckesnetzen auf die Hälfte. Viereckesstreifen beziehungsweise Dreicksstreifen sind unter anderem Bestandteil von OpenGL [SGIc] und werden oftmals direkt durch die Hardware unterstützt. Eine weitere Optimierung würde einen Support der genannten Applikationen durch die Hardware zu aufwändig gestalten. Unglücklicherweise ist auch hier noch viel Redundanz vonnöten, da zwei adjazente Streifen die entsprechende Reihe von Scheitelpunkten doppelt speichern.

4.1.4 Parametrische Flächen und Kurven

Die in den Polygonnetzen verwendeten Polygone stellen stückweise lineare Approximationen für Kurven und Flächen dar. Polygonnetze kennzeichnen sich daher durch eine hohe Zahl

an Scheitelpunkten aus, welche für möglichst genaue Repräsentationen erforderlich ist. Eine kompaktere Darstellung basiert auf der Verwendung sogenannter Freiformflächen oder -kurven, insbesondere auf stückweisen polynomialen Flächen beziehungsweise Kurven. Im Bereich des Computer Aided Designs werden Freiformflächen ungefähr seit dem Jahr 1958 eingesetzt, wobei der damalige Schwerpunkt vor allem im Automobil-Karosseriebau lag. Aus diesem Grund ist es nicht verwunderlich, dass zwei der Pioniere, nämlich Pierre Bézier und Paul de Casteljau, für die bekannten Automobil-Konzerne Renault beziehungsweise Citroen tätig waren.

Grundsätzlich ist zwischen der expliziten, der impliziten und der parametrischen Darstellung von Flächen oder Kurven zu unterscheiden. In der expliziten Form $y = f(x)$ darf es für jeden Wert x auch nur jeweils einen Wert y geben, weshalb beispielsweise ein Kreis oder eine Ellipse nicht geschlossen, sondern lediglich mit Hilfe mehrerer Segmente dargestellt werden kann. Zudem ist die explizite Beschreibung nicht invariant gegenüber Rotationen und kann keine Kurven mit wirklich vertikalen Tangenten repräsentieren. Bei der impliziten Darstellung $f(x, y) = 0$ besteht die Möglichkeit mehrerer Lösungen, weshalb für ein eindeutiges Ergebnis zusätzliche Randbedingungen erstellt werden müssen. Ein weiteres Problem der impliziten Darstellung ist die oftmals nicht triviale Bestimmung der Tangenten. Abschnitt 4.1.5 gibt einen genaueren Überblick auf die implizite Vorgehensweise.

Gegenüber der expliziten und der impliziten Form weist die parametrische Darstellung einer Kurve $Q(u) = (X(u), Y(u))$ mehrere Vorteile auf[1]: Zum einen gibt es keine mehrdeutigen Lösungen und zum anderen werden geometrische Steigungen durch Tangentenvektoren ersetzt, welche im Gegensatz zu den Steigungen selbst niemals unendlich sein können. Ein weiterer Vorteil liegt in der Invarianz gegenüber Rotationen. Parametrische Kurven $Q(u) = p_0 + p_1 u + p_2 u^2 + \ldots + p_k u^k$ sind in der Regel durch ganz rationale oder gebrochen rationale Polynome n-ten Grades bestimmt, wobei in der Computer Grafik überwiegend kubische Repräsentationen mit $k = 3$ genutzt werden. Diese stellen die niedrigste Ordnung da, um eine Kurve über zwei Endpunkte und deren Steigung zu spezifizieren. Während Polynome niederer Ordnung oftmals nicht die gewünschte Flexibilität bei der Modellierung bieten und zudem koplanar im 3D Raum sind, ist die Verwendung von Polynomen höherer Ordnung wesentlich rechenintensiver und führt unter Umständen zum Problem der Oszillation.

Die parametrische Repräsentation von Kurven und Flächen gliedert sich in exakte, interpolatorische und approximative Darstellungen. Bei der exakten Darstellung ist jeder Punkt durch eine Formel definiert, wobei letztere jedoch zumeist entweder nicht bekannt oder zu komplex ist. Die interpolatorische Darstellung beschreibt ebenso wie die approximative Form eine Fläche oder Kurve durch Stützstellenbedingungen. Im Gegensatz zur interpolativen Repräsentation determiniert in der approximativen Form die Fläche oder Kurve jedoch nicht an den Stützstellen. Für beide Ansätze existiert eine ganze Reihe von Vorgehensweisen. Beispiele für Interpolationsschemata sind die Interpolation mit Monomen sowie die Interpolationen

[1] Die hier aufgeführten Gleichungen beschreiben lediglich Kurven im 2D Raum. Für die Verwendung im 3D Raum muss die explizite Form um $z = g(x)$ und die implizite Form auf $f(x, y, z) = 0$ erweitert werden. Die Beschreibung einer parametrischen Fläche im 3D Raum ist mit der Gleichung $Q(u, v) = (X(u, v), Y(u, v), Z(u, v))$ gegeben. Üblicherweise ist das Intervall der Parameter u und v auf $0 \leq u \leq 1$ beziehungsweise $0 \leq v \leq 1$ beschränkt. Komplexe Modelle setzen sich daher aus mehreren parametrischen Beschreibungen, sogenannten Segmenten oder Patches, zusammen.

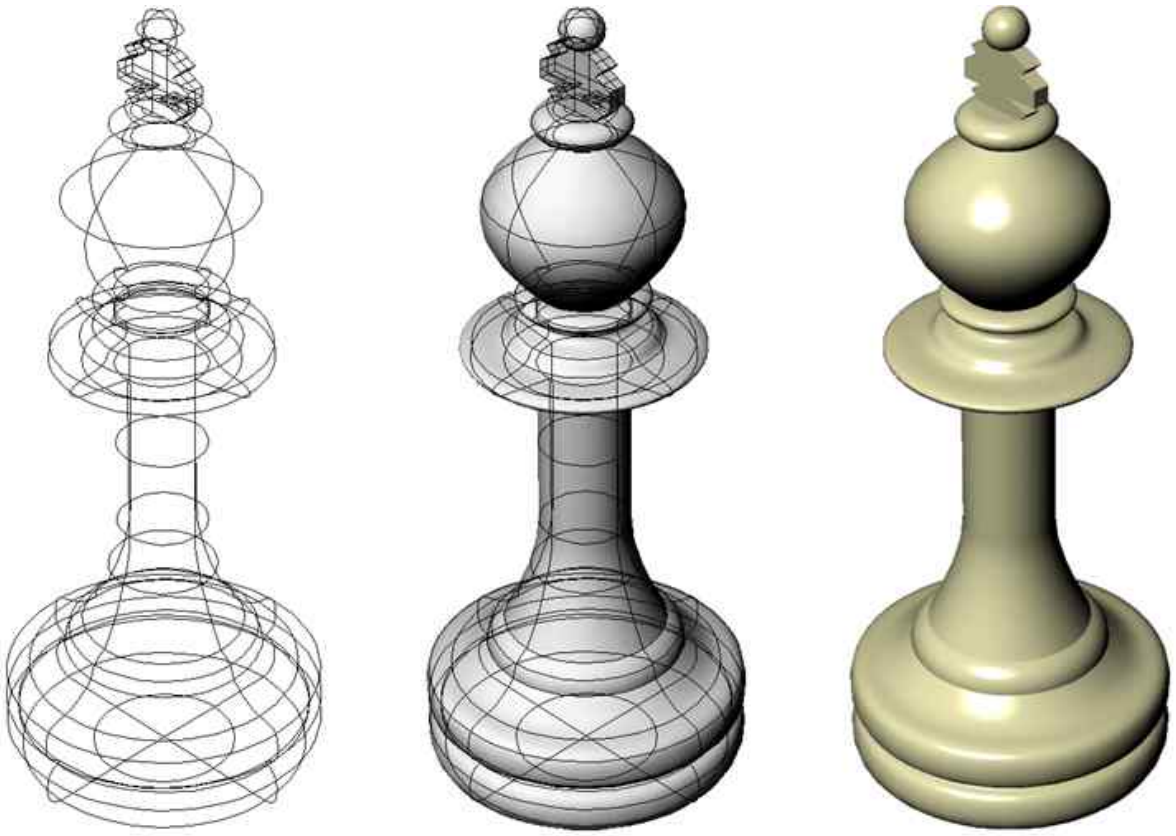

Abbildung 4.2: *Eine Schachfigur modelliert über NURBS Patches.*

mit Newton-Polynomen, Lagrange-Polynomen oder Tschebyscheff-Polynomen [2]. Unabhängig von der Methode hat die Interpolation immer das bereits erwähnte Problem der Oszillation, insbesondere bei hohem Polynomgrad. Approximationen vermeiden zwar Oszillationen, allerdings verläuft die resultierende Kurve nicht notwendigerweise durch die gewählten Stützstellen, weshalb das Aussehen der Kurve im Rahmen eines interaktiven Modellierungsprozesses schwieriger vorhersagbar ist. Bedeutende Repräsentationen sind die Bézier-Kurven [B́72] mit Hilfe von Bernstein-Polynomen [For72], Hermite-Kurven [BS70] sowie die Formulierung von de Casteljau [Cd59].

Eine weitere wichtige approximative Beschreibungsform sind Splines, welche das mathematische Äquivalent zu den physikalischen Splines (im Deutschen auch Straklatten) darstellen. Der englische Begriff Spline steht für einen dünnen Stab, der von Schiffsbauern verwendet wurde, um die richtige Form der Planken zu bestimmen, welche quer zu den Spanten die Außenwand des Schiffsrumpfes bilden. Kubische Basis-Splines oder auch B-Splines bieten per Definition eine C^2 Stetigkeit an Segmentübergängen und verlaufen dort somit glatter als Hermite- oder Bézier-Kurven. Die parametrische Stetigkeit C^n verlangt dabei, dass die n-te Ableitung der Kurvensegmente am Verknüpfungspunkt identisch ist. B-Splines offerieren einfache Möglichkeiten zur interaktiven Manipulation, da die Kurvensegemente nur von

[2] Für eine Übersicht siehe [HL92].

wenigen Kontrollpunkten abhängig sind und somit eine lokale Kontrolle erlauben. Es gibt viele weitere Arten von Splines wie etwa Catmull-Rom-Splines oder β-Splines. Eine herausragende Stellung nehmen die Nonuniform Rational B-Splines (NURBS) ein, weil sie nicht nur invariant bezüglich Translation, Rotation und Skalierung sind, sondern auch bezüglich perspektivischen Transformationen. Die rationalen Funktionen werden dabei als Quotient zweier Polynome definiert:

$$x(u) = \frac{X(u)}{W(u)}, y(u) = \frac{Y(u)}{W(u)}, z(u) = \frac{Z(u)}{W(u)} \tag{4.8}$$

Diese Vorgehensweise kann mit $Q(u) = (X(u), Y(u), Z(u), W(u))$ als eine Definition der Kurve im homogenen Raum interpretiert werden, wobei jede polynomiale Kurve durch Hinzufügen von $W(u) = 1$ zu einer rationalen Kurve wird.

4.1.5 Implizite Flächen

Die Klasse der impliziten Flächen ist größer als die der parametrischen Flächen, da letztere zwar in implizite Flächen umgewandelt werden können, aber nicht umgekehrt [SAG84]. Implizite Flächen oder auch Isoflächen bilden die Grundbausteine für komplexe implizite Geometrien. Ihr Ziel ist die Modellierung geschlossener, glatter Elemente wie etwa Sphären. Eine implizite Geometrie wird definiert durch eine Basisgeometrie B, eine Distanzfunktion $d_b(P)$ und einen Schwellwert beziehungsweise Isowert $s > 0$. Im dreidimensionalen Raum R^3 ist die implizite Geometrie durch die Menge M_s der Punkte $P \in R^3$ gegeben, für die $d_b(P) = s$ ist. Tabelle 4.2 führt als Beispiel die Definition einer Kapsel über die Basisgeometrie einer Strecke L an. Abbildung 4.3 skizziert die Beschreibung im zweidimensionalen Raum R^2. Anhand der Illustration wird deutlich, dass es sich bei der mathematischen Definition um eine durchaus intuitive und logische Vorgehensweise handelt. Es ist sicherlich für einen Menschen auch kein Problem, die Visualisierung der Beschreibung wie in Abbildung 4.3 auf Papier zu bringen. Doch wie kann eine Visualisierung auf einem Computer realisiert werden? Im Falle der Kapsel wäre dies noch durch eine einfache Segmentierung in die beiden Kopfenden und dem Mittelteil, in dem $\overline{P_L(P)P}$ senkrecht zur Strecke L gilt, möglich. Allerdings wird eine allgemeine Lösung für den dreidimensionalen Raum benötigt.

Implizite Flächen	**Kapsel**
Basisgeometrie B	Strecke L $L(\alpha) = V + \alpha\vec{I}, \alpha \in [0,1]$
Distanzfunktion $d_b(P)$ Abstand von P zur Basisgeometrie	$d_L(P) = min\{\|P - Q\| : Q \in L\}$ Abstand von P zur Strecke L
Schwellwert/Isowert $s>0$	$s>0$
Geometrie: $M_s = \{P \in R^3 : d_b(P) = s\}$	Kapsel: $M_s = \{P \in R^3 : d_L(P) = s\}$

Tabelle 4.2: *Die Definition einer Kapsel mit Hilfe impliziter Flächen.*

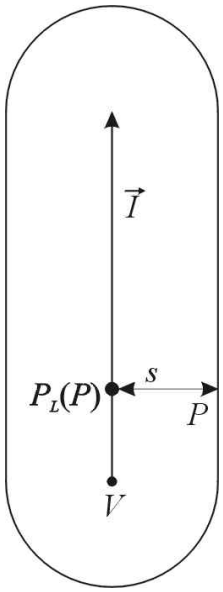

Abbildung 4.3: *Die Definition einer Kapsel über die Basisgeometrie der Strecke L.*

Zwar existiert mit dem Marching Cube Algorithmus eine entsprechende Lösung, doch ist das Verfahren für eine Echtzeitberechnung vieler Elemente nicht geeignet.

Der Marching Cube Algorithmus lässt das Problem der impliziten Flächen erahnen, vor der Visualisierung erst eine Polygonalisierung vornehmen zu müssen, d.h. ein über implizite Flächen berechnetes Modell erfordert bedingt durch die Grafik-Hardware die Transformation in ein Mesh. Somit ist der Algorithmus bei den Leistungsmerkmalen derzeitiger Personal Computer für eine Echtzeitberechnung vieler Elemente nicht einsetzbar. Zudem ist es unbefriedigend, auf der einen Seite eine exakte Modellierung vorzunehmen und auf der anderen Seite doch wieder nur eine approximierte Visualisierung zu erhalten, ein Problem, welches aufgrund der Rasterisierung für alle Kurvenmodellierungen oder Freiformbeschreibungen gilt.

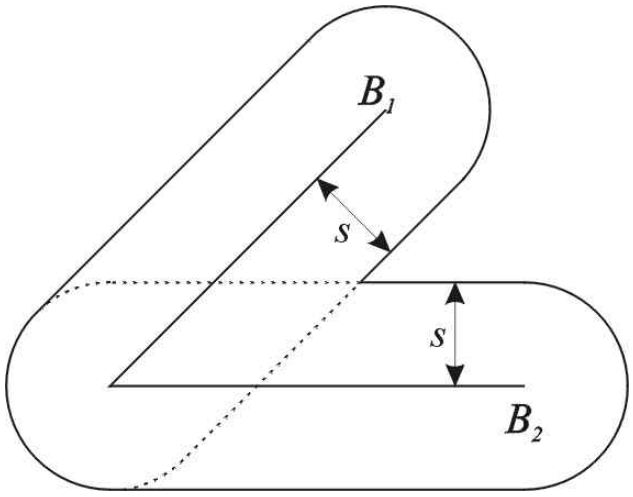

Abbildung 4.4: *Der Ansatz des minimalen Abstands verwendet für verschiedene Basisgeometrien den gleichen Schwellwert s.*

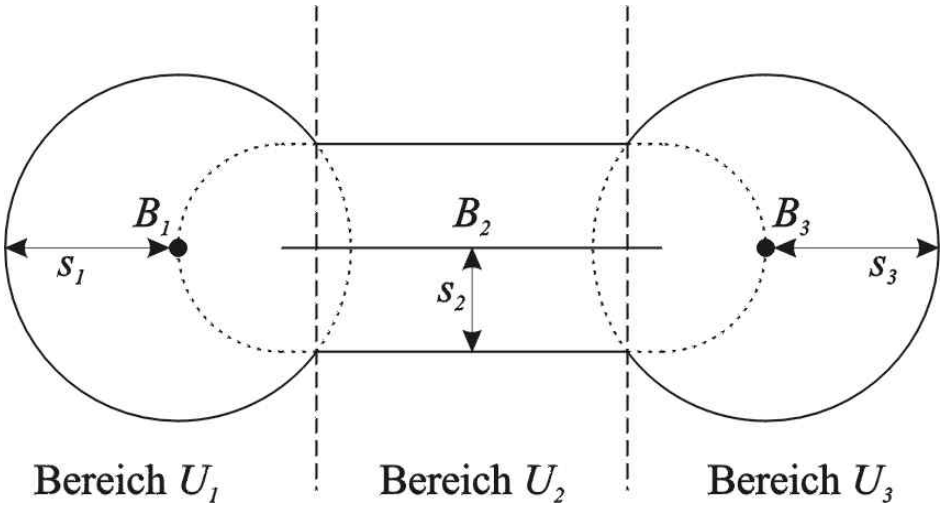

Abbildung 4.5: *Der Raumunterteilungsansatz definiert die Relevanz einer Abstandsfunktion mit Hilfe einer räumlichen Zerlegung.*

Eine andere Problematik impliziter Flächen betrifft die zum Teil nicht triviale Bestimmung von Flächennormalen für die Berechnung der Abstandsfunktion.

Trotz dieser Schwierigkeiten stellen implizite Flächen eine ansprechende Lösung dar, um bestimmte Elemente effizient zu modellieren. So bieten sie zum Teil einfache Schnitt- und Abstandsberechnungen, weshalb sie beispielsweise als Bounding Volumes in Kollisionserkennungen eingesetzt werden können. Zudem erlauben sie mit Hilfe des Marching Cube Algorithmus eine automatische Generierung von Meshes mit verschiedenen Auflösungen. Das Design einer runden Oberfläche über ein von Hand erzeugtes Meshes impliziert für den betreffenden nicht nur sehr viel Aufwand, sondern auch ein gutes Nervenkostüm und Fingerspitzengefühl. Eine weitere Möglichkeit zur Modellierung ist über die Verknüpfung einfacher impliziter Geometrien zu komplexen Geometrien gegeben. Mit dem Ansatz des minimalen Abstands und dem Ansatz der Raumaufteilung existieren zwei derartige Vorgehensweisen, die beide auf einer Basisgeometrie bestehend aus mehreren Teilen $B_1, \ldots, B_k$ beruhen, wobei k die maximale Anzahl der Geometrien vorgibt. Das Verfahren des minimalen Abstands bestimmt ausgehend von einem Punkt $P \in R^3$ die relevante Abstandsfunktion über $d_B(P) = min\{d_{B_i}(P) : i = 1, \ldots, k\}$. P ist Teil der Geometrie falls $d_B(P)$ dem in Bezug auf alle $B_1, \ldots, B_k$ konstanten Schwellwert s entspricht.

Im Gegensatz dazu besteht die Vorgabe des Raumunterteilungsansatzes entsprechend der Basisgeometrien $B_1, \ldots, B_k$ aus $s_1, \ldots, s_k$ Schwellwerten. Zusätzlich erfolgt eine disjunkte räumliche Zerlegung $R^3 = \dot{\bigcup}_{i=1}^{k} U_i$ in k Bereiche $U_1, \ldots, U_k$. Analog zum Verfahren des minimalen Abstands wird auch hier über einen Punkt $P \in R^3$ die relevante Abstandsfunktion bestimmt, allerdings in Abhängigkeit von dem Bereich U_i, in dem sich P befindet. Somit gilt $d_B(P) = d_{B_i}(P) \Leftrightarrow P \in U_i$. Unter dieser Bedingung ist P Teil der Geometrie wenn $d_B(P) = s_i$.

4.2 Animationselemente

In den vorangegangen Abschnitten wurden Techniken zur Modellierung der Elemente einer Szene vorgestellt. Insbsondere komplexe Elemente verfügen jedoch nicht nur über visuelle Eigenschaften, sondern zusätzlich über bestimmte Verhaltensmerkmale. Luckas stellt in [Luc00] ein Konzept vor, mit dem die aus dem Bereich der 2D Grafik bekannte Clipart Metapher auf 3D Animationen übertragen wird. Eine Animation wird danach durch das Zusammenstellen von vordefinierten Animationselementen erzeugt. Im Gegensatz zu bereits existierenden 3D Elementbibliotheken beinhalten diese Bausteine nicht nur 3D Geometrien, sondern auch Methoden zu deren Animation. Dies umfasst sowohl allgemeine Animationsmethoden wie das Platzieren oder das Verschieben eines Elements als auch elementspezifische Methoden wie etwa das Öffnen einer Tür bei einem Auto. Mit diesem Ansatz soll die Generierung einer 3D Animation wesentlich vereinfacht und beschleunigt werden.

Luckas verwendet dabei seine Definition aus [LD98], wonach ein Animationselement als ein unabhängiger, gekapselter Baustein definiert wird, der zur Erzeugung dynamischer Szenarien verwendet wird. Er repräsentiert ein Objekt der Welt, welche der Benutzer innerhalb der animierten Szene modellieren will. Das Animationselement beinhaltet sowohl die Beschreibung der visuellen Erscheinungsform des korrespondierenden Objekts (Geometrie, Farbe, Material, Texturen, ...) als auch Wissen über dessen spezifisches Verhalten (Definition der Animation, Anwendungslogik, Interaktionsmöglichkeiten mit anderen Animationselementen oder einem Benutzer).

In der von Luckas präsentierten Architektur folgt das Konzept der Animationselemente dem Prinzip der Objektorientierung. Animationselemente sind demnach Objekte, die als Instanzen von Klassen gebildet werden. Die Funktionalität zur Änderung der eigenen Erscheinung und des inneren Zustands sowie zur Interaktion und Kommunikation mit anderen Animationselementen, kurz das Verhalten, wird in einem Animationselement gekapselt. Aufgrund Anforderung 2.2, welche unter anderem die mögliche Heterogenität der Endgeräte verlangt, treten an dieser Stelle allerdings Probleme auf. In einem objektorientierten Ansatz müsste ein Animationselement beispielsweise auch Methoden für die Visualisierung auf allen potentiellen Endgeräten zur Verfügung stellen. Werden zusätzlich noch verschiedene Visualisierungsformen wie etwa Wireframe- oder Texturdarstellungen erwartet, so steigt die Anzahl der erforderlichen Methoden zu einer unübersichtlichen Menge an. Das ist umso ärgerlicher falls auf einem Endgerät nur ein sehr geringer Anteil der Visualisierungsmethoden tatsächlich verwendet wird. Somit ist bereits an dieser Stelle ersichtlich, dass die Anforderungen aus Kapitel 2 ein flexibleres Konzept für eine Realisierung der Animationselemente nötig machen.

4.3 Level of Detail

Trotz der immer weiter steigenden Leistungsmermale derzeitiger Endgeräte sind auch heute noch die Auflösungen der Polygonnetze für die Repräsentation der Elemente einer Szene sehr stark eingeschränkt. Beispielsweise verfügen die Figuren in aktuellen 3D Spielen selbst im

höchsten Detaillierungsgrad über nicht mehr als 2000-3000 Polygone, um interaktive Frameraten zu erreichen. Entsprechend hölzern wirken dann auch die Gesichtsausdrucke der virtuellen Charaktere. Aus diesem Grund wird mit Hilfe von Techniken wie Texturen, Licht-Schatten-Berechnungen oder Shadern versucht, die geringe Zahl der Polygone zu kaschieren und eine nicht vorhandene Komplexität vorzutäuschen. Somit tragen die Polygonnetze nicht die alleinige Verantwortung für das visuelle Erscheinungsbild, sondern auch die korrespondierenden Zusatzinformationen wie Farbwerte, Normalen oder Texturen einschließlich der verwendeten Visualisierungsmethoden haben ihren Anteil. Da jedoch nicht nur die computergestützte Visualisierung, sondern auch die optische Wahrnehmung des Menschen sowohl in räumlicher als auch in zeitlicher Auflösung beschränkt ist, stellt sich die Frage, wann sich denn überhaupt der Einsatz einer bestimmten Technologie oder Auflösung lohnt. Hierzu empfahl Clark bereits 1976 den Einsatz vereinfachter Versionen von visuell weniger relevanten Elementen [Cla76] und bezeichnete diese Versionen als *Level-of-Detail* (LOD). Was aber sind mögliche Kriterien, um visuell weniger relevante Elemente zu identifizieren beziehungsweise das Ausmaß ihrer Relevanz zu bestimmen? Die nachfolgende Übersicht stellt einige übliche Ansatzpunkte vor.

- **Entfernung:** Aufgrund der perspektivischen Wahrnehmung der Menschen erscheinen vom Betrachter weit entfernte Elemente sehr viel kleiner als von einem nahen Betrachterstandpunkt. Da analog für die computergestützte Visualisierung perspektivische Transformationen verwendet werden, ergibt sich hier eine ähnliche Situation. Ein im Hintergrund liegendes Element beansprucht weniger Pixel als eine Rasterisierung desselben direkt vor dem Beobachter. Wenn ein Element aber nur durch wenige Pixel repräsentiert ist, dann ist auch eine hohe Qualität der Polygonnetze überflüssig, da die Details ohnehin in der diskreten Auflösung der Grafik-Hardware verloren gehen. Entsprechend kann beispielsweise auch der Einsatz einer einfarbig gefüllten Fläche an Stelle einer Textur durchaus sinnvoll sein. Eine effiziente Berechnung dieses Kriteriums ist über das Quadrat der *Euklidischen Distanz* möglich, wodurch die rechenintensive Bestimmung der Wurzel überflüssig wird.

- **Fläche:** Dieses Kriterium ist ähnlich zu dem vorangehenden, allerdings wird hier von der wirklichen Größe des Elementes und nicht von der Entfernung ausgegangen. Kleine Elemente belegen auch dann nur eine geringe Pixelmenge, wenn sie direkt vor dem Betrachter positioniert sind. Somit müssen kleine Elemente nicht über den gleichen Detaillierungsgrad wie größere verfügen. Die Berechnung dieses Kriteriums gestaltet sich aufwändiger als im Falle der Entfernung, da die Menge der beanspruchten Pixel eines Elementes zumindest näherungsweise bestimmt werden muss.

- **Fokus:** Elementen im zentralen Sichtbereich kommt eine höher Bedeutung zu als Elementen, die sich am Rand des Sichtbereiches befinden. Dieser Aspekt ist unmittelbar mit der Dichte von Stäbchen und Zäpfchen auf der menschlichen Netzhaut (Retina) verknüpft. Eine Bestimmung dieses Kriteriums erfordert die Position des Elementes auf dem Bildschirm, was zwar auf dem ersten Blick wie ein Nebenprodukt der Visualisierung wirkt, aber in Wirklichkeit nicht immer ist. Bei der Verwendung von OpenGL beispielsweise definiert der Benutzer ein Element in Objektkoordinaten und legt daraufhin

Abbildung 4.6: *Einige Spiele wie Prince of Persia 3 verwenden Bewegungsunschärfen, um den Ablauf einer Animation zu verdeutlichen (Screenshot aus Prince of Persia 3 [PRI]).*

die entsprechenden Matrizen für die Transformationen in Weltkoordinaten beziehungsweise für die perspektivische Transformation fest. Der Visualisierungsprozess erfolgt für den Benutzer vollständig transparent, d.h. er hat keine Möglichkeit, die Ergebnisse der einzelnen Transformationsstufen abzurufen. Somit muss hier die Berechnung der Bildschirmposition eines Elementes redundant erfolgen, was einen signifikanten Mehraufwand bedeuten kann. Die Identifikation von Randelementen anhand der durch die Navigation transformierten Sichtpyramide des Betrachters direkt im dreidimensionalen Raum vorzunehmen, würde im Vergleich zur Abbildung in zweidimensionale Bildpositionen wohl noch mehr Rechenkapazität erfordern.

- **Dynamik:** Dynamische, insbesondere sich schnell bewegende Elemente erscheinen aufgrund der zeitlich begrenzten Auflösung des menschlichen Auges verschwommen. Dieses Phänomen wird auch als Bewegungsunschärfe bezeichnet und bietet vor allem die Möglichkeit, den Aufwand für bestimmte Technologien einzusparen. Beispielsweise sind Verfahren wie die äußerst rechenintensive Kantenglättung (Antialiasing) während einer schnellen Navigation des Betrachters durch eine Szene überflüssig. Es gibt jedoch mittlerweile Spiele wie etwa Prince of Persia, welche Beweungsunschärfen bewusst darstellen, um die Schnelligkeit und Richtung einer Bewegung zu betonen (siehe Abbildung 4.6).

Eines der am häufigsten eingesetzten Kriterien ist aufgrund seiner einfachen und schnellen Berechnung die Entfernung eines Elementes zum Betrachter. Da die Entfernung ein konti-

nuierliches Kriterium darstellt, wird ausgehend von der maximalen Sichtweite eine Diskretisierung in mehrere Intervalle vorgenommen und jedem Intervall eine Detailstufe zugeordnet. Bei der Umschaltung zwischen zwei Stufen können sogenannte Plop-Effekte auftreten, d.h. ein Element, welches eben noch als schlichter Punkt am Horizont zu sehen war, wächst auf einmal sprunghaft an und erscheint in einem wesentlich höheren Detaillierungsgrad. Ziel sollte daher ein möglichst kontinuierlicher Übergang zwischen den einzelnen Stufen sein, was zwangsläufig eine feine Diskretisierung und somit eine entsprechende Anzahl von Detailstufen des gleichen Elementes voraussetzt. Da eine automatische Berechnung der Level-of-Detail mit Hilfe derzeitiger Algorithmen in Echtzeit und ansprechender visueller Qualität nicht möglich ist, werden die einzelnen Detailstufen in einem Vorverarbeitungsschritt generiert und konserviert. Somit verwaltet die spätere Applikation nicht nur ein Modell, das sie je nach den gegebenen Kriterien transformiert, sondern sie hält sämtliche Versionen eines Elements im Speicher, die mit hoher Wahrscheinlichkeit auch noch über redundante Informationen verfügen. Das Konzept der Vorberechnung mehrerer Versionen eines Elements mit unterschiedlichen Detaillierungsgraden ist auch unter dem Begriff *Multi-Resolution* bekannt.

Obgleich also mit Hilfe vereinfachter Darstellungen der Visualisierungsaufwand gesenkt wird, ist als Nebeneffekt der erhöhte Speicherverbrauch zu berücksichtigen. Dies gilt insbesondere für dynamische Elemente, bei denen aufgrund einer Animation für alle Teilschritte der Bewegung das jeweils resultierende Modell in verschiedenen Level-of-Detail generiert werden muss. Derartige Beispiele finden sich häufig in 3D Spielen, da nicht nur die automatische Berechnung der Detailstufen, sondern ebenfalls die Bestimmung komplexer Animationen beispielsweise mit Hilfe von Skelettanimationen oder inverser Kinematik für mehrere Elemente noch nicht in Echtzeit möglich ist. Wegen des begrenzten Speicherplatzes erfolgt somit in der Regel ein Kompromiss zwischen der Qualität der Animation und der Qualität der Level-of-Detail.

Obwohl aufgrund der Vorberechnung der zeitliche Aufwand für die automatische Generierung der Detailstufen nur noch bedingt eine Rolle spielt, ist es auch heute noch in der Spielebranche üblich, die Elemente eines Spiels über einen teuren Designprozess in den gewünschten Stufen, Perspektiven und Animationsschritten von Hand zu entwerfen. Die Ursache liegt im optimalen Kompromiss zwischen dem visuellen Eindruck einer Detailstufe und der in dieser Stufe möglichen Komplexität der Daten. Auch wenn zwei Polygonnetze über die gleiche Anzahl an Polygonen verfügen, so kann die Approximation an das repräsentierte Modell doch sehr unterschiedlich ausfallen. Wie soll beispielsweise ein Algorithmus im Falle der automatischen Generierung von Detailstufen für einen Mercedes-Benz feststellen, dass der Mercedes-Stern für die Wiedererkennung des Automobils von entscheidender Bedeutung ist und deshalb nicht der Vereinfachung zum Opfer fallen sollte? Hier können Algorithmen zwar unterstützend eingesetzt werden, jedoch sind sie, etwa durch die Markierung des Mercedes-Sterns, auf die Interaktion mit dem Benutzer angewiesen.

Trotz dieser Problematik wurden in den letzten Jahren verschiedene Technologien zur automatischen Generierung von Level-of-Detail entwickelt, die sich nach [Eri96] in Verfahren zur Entfernung von Geometrie (Decimation, Simplifizierung), in (Re-) Sampling Ansätze und in Verfeinerungsalgorithmen (Refinement, Subdivision) klassifizieren lassen. In der Vielzahl dieser Algorithmen gibt es keinen universellen Ansatz, der auf jede Frage die richtige Antwort parat hält. Vielmehr muss basierend auf den Vorgaben und dem gewünschten Endergebnis

eine Auswahl getroffen werden. Hierbei sind grundsätzliche Kriterien zu berücksichtigen wie etwa die Laufzeit oder die Einbeziehung von Zusatzattributen. Eine Vereinfachung als invasives Verfahren führt immer zum Verlust oder zur Modifizierung von Informationen. Wenn zum Beispiel ein Vertex aus einem Mesh entfernt wird, so stellt sich die Frage, was mit den korrespondierenden Texturkoordinaten oder Farbwerten geschieht. Fallen sie ebenfalls weg? Müssen sie vielleicht über Nachbarschaftsbeziehungen interpoliert werden? Derartige Aspekte sind sehr wichtig, wenn sich die Visualisierung nicht nur auf Drahtgittermodelle beschränken soll.

4.3.1 Entfernung von Geometrie

Verfahren dieser Kategorie beruhen auf der Vereinfachung des Originalmeshes $\hat{M} = M^n$ in das nicht weiter zu simplifizierende Basismesh M^0. Dazu werden n Transformationen benötigt, die jeweils den Übergang eines Meshes M^i in das vereinfachte Mesh M^{i-1} beschreiben. Jede Transformation wird durch einen Operator definiert, der auf das Polygonnetz angewendet wird. Dabei ist zwischen topologischen und geometrischen Operatoren zu unterscheiden. Erstere können sowohl Auswirkungen auf die lokale als auch auf die globale Topologie eines Meshes haben. Eine Veränderung der lokalen Topologie betrifft die unmittelbare Umgebung des transformierten Bereiches. Sie kann aber muss nicht zwangsläufig die globale Topologie beeinflussen, wie es durch eine Manipulation des Geschlechts oder der Anzahl der Zusammenhangskomponenten der Fall wäre. Im Gegensatz zu topologischen Operatoren nehmen geometrische Operatoren keine Änderungen an der Topologie vor, sondern verschieben lediglich die Position eines Scheitelpunktes. Derartige Operatoren führen zwar nicht zu einer Vereinfachung des Meshes, können aber durch eine bessere Approximation an das Originalmesh eine Steigerung seiner Qualität bedeuten. Häufig werden auch hybride Operatoren verwendet, welche sowohl über geometrische als auch topologische Eigenschaften verfügen. Da fast alle Algorithmen Dreiecksnetze voraussetzen, bezieht sich die folgende Auflistung möglicher Transformationen ausschließlich auf diese spezialisierte Form der Polygonnetze.

- **Vertex Change:** Der *Vertex Change Operator* ist ein rein geometrischer Operator, der lediglich die Position eines Scheitelpunktes verschiebt, um eine bessere Approximation des aktuellen Meshes M^i mit $0 \leq i < n$ an das vorangehende Mesh M^{i+1} zu erreichen.

- **Half-Edge Collapse:** Der *Half-Edge Operator* entfernt eine Kante aus einem Mesh und fasst die beiden korrespondierenden Scheitelpunkte zu einem einzelnen Scheitelpunkt zusammen. Dieser Operator ist rein topologisch, da der verwendete Scheitelpunkt einem der beiden Vertices der Kante entspricht.

- **Edge Collapse:** Der *Edge Collapse Operator* ist eine Kombination aus Vertex Change Operator und Half-Edge Collapse Operator. Zwar wird auch hier eine Kante kollabiert, aber anstatt einen der beiden korrespondierenden Scheitelpunkte zu verwenden, werden diese per Vertex Change Operator zu einem neuen Vertex zwischen den ursprünglichen Positionen zusammengefasst. Somit ist der Edge Collapse Operator eine sowohl topologische als auch geometrische Vorgehensweise.

- **Edge Split:** Der *Edge Split Operator* unterteilt eine Kante durch das Einfügen eines neuen Scheitelpunktes. Die zu der Kante adjazenten Dreiecke werden dem neuen Vertex entsprechend umstrukturiert beziehungsweise ebenfalls unterteilt. Ebenso wie sein Vorgänger ist der Edge Split Operator hybrid, d.h. Topologie und Geometrie werden beide verändert.

- **Edge Swap:** Der *Edge Swap Operator* verändert Start- und Endpunkte einer Kante, in dem er ihr andere Vertices der adjazenten Polygone zuweist. Dabei wird nur die Topologie manipuliert.

- **Vertex Removal:** Der *Vertex Removal Operator* entfernt einen Scheitelpunkt aus dem Dreiecksnetz und definiert eine neue Triangulisierung der lokalen Topologie. Zwar ist die Reduktion des Speicheraufwands analog zum Half-Edge Operator, jedoch kann das Mesh ähnlich dem Vertex Change Operator aufgrund anderer Kriterien verbessert werden. Ein Beispiel ist eine neue Triangulisierung der lokalen Topologie in möglichst gleichgroße Dreiecke. Der Vertex Removal Operator verfügt lediglich über topologische Eigenschaften.

- **Vertex Contract:** Der *Vertex Contract Operator* fasst zwei nicht benachbarte, d.h. nicht über eine Kante verbundene Scheitelpunkte zu einem neuen Scheitelpunkt zusammen. Hierüber sind im Gegensatz zum Edge Collapse Operator auch Veränderungen der globalen Topologie möglich, da beispielsweise Zusammenhangskomponenten verbunden werden können. Der Vertex Contract Operator ist ein hybrider Operator.

- **Vertex Split:** Der *Vertex Split Operator* dient als Umkehroperator zum Edge Collapse Operator beziehungsweise zum Half-Edge Collapse Operator. Im ersten Fall werden für einen Scheitelpunkt v zwei neue Scheitelpunkte v_1 und v_2 eingefügt, im zweiten Fall dagegen gilt $v_1 = v \vee v_2 = v$. Der Vertex Split Operator wird in der Regel für die Verfeinerung eines Polygonnetzes M^i in das Mesh M^{i+1} mit $0 \leq i < n$ eingesetzt. Somit sind v_1 und v_2 Teil der Knotenmenge des Netzes M^{i+1} und müssen nicht neu berechnet, sondern nur aus einer entsprechenden Datenstruktur geladen werden. In diesem Fall handelt es sich um einen topologischen Operator.

Wie in Abbildung 4.7 zu sehen, wird bei der Simplifizierung immer eine *Wedge* beziehungsweise ein *Keil* des Polygonnetzes betrachtet. Darunter ist ein Scheitelpunkt und seine ihn indizierenden Dreiecke zu verstehen. Vor der Anwendung eines Operator muss in vielen Fällen die Korrektheit des Keils überprüft werden, um beispielsweise die Vereinfachung von Rändern eines Polygonnetzes zu vermeiden. Eine Klassifizierung verschiedener Keilarten findet sich unter anderem bei Schröder et al. [Sch97]. Die entsprechende Arbeit wird aber noch im späteren Verlauf des Abschnitts vorgestellt.

Die Qualität eines transformierten Meshes wird über ein Qualitätsmaß definiert, das die Güte der Approximation an das Originalmesh $\hat{M}$ beschreibt. Wichtige Begriffe sind in diesem Zusammenhang die lokale und die globale Fehlermetrik. Mit Hilfe der lokalen Fehlermetrik wird die Differenz beziehungsweise der Fehler bestimmt, der durch die Transformation eines Meshes M^i in das vereinfachte Mesh M^{i-1} mit $0 < i \leq n$ entsteht. Als Beispiel für eine lokale Fehlermetrik kann der häufig verwendete Einseitige Hausdorff-Abstand angeführt

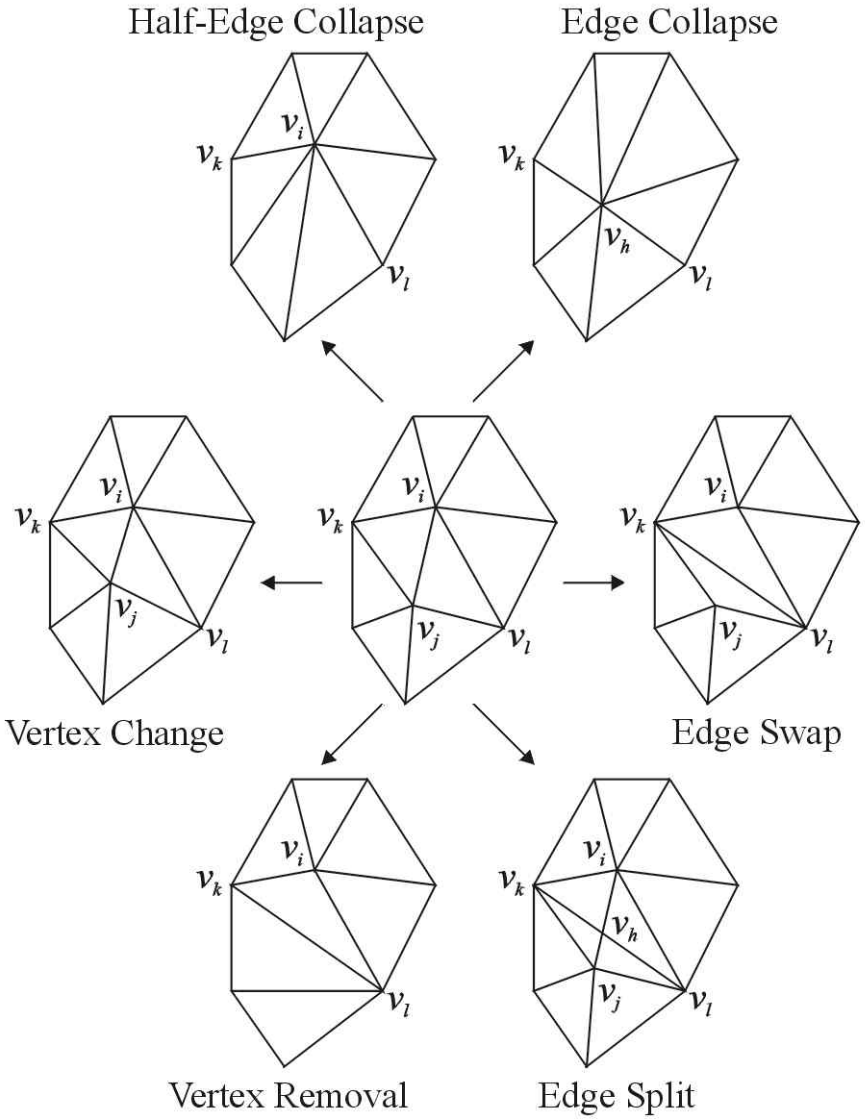

Abbildung 4.7: *Die Operatoren dienen zur Tranformation eines Polygonnetzes. Das Resultat kann aus einem vereinfachten, verfeinerten oder qualitativ hochwertigeren Polygonnetz bestehen. Illustrationen des Vertex Contract Operators und des Vertex Split Operators befinden sich in Abbildung 4.8 beziehungsweise 4.9.*

werden, welcher den Abstand eines Scheitelpunktes v im Mesh M^i zu einer durch das Entfernen von v entstehenden Fläche im Mesh M^{i-1} bestimmt. Mögliche Operatoren für einen Einsatz des einseitigen Hausdorff-Abstands sind unter anderem der Half-Edge Collapse Operator, der Edge Collapse Operator oder der Vertex Removal Operator. Im Unterschied zur lokalen Fehlermetrik beschreibt die globale Fehlermetrik die Differenz eines Meshes M^i mit $0 \leq i < n$ zum Originalmesh $\hat{M}$. Da eine absolute Berechnung der globalen Fehlermetrik für ein bestimmtes Mesh M^i sehr aufwändig ist, wird sie häufig über die Akkumulation der lokalen Fehler angenähert. Überschreitet dieser Wert eine vorgegebene Fehlerschranke, so wird der Algorithmus beendet. Ein andere Vorgehensweise verwendet Simplification Envelopes, mit deren Hilfe um das gesamte Originalmesh sowohl nach innen als auch nach außen eine Distanz beschrieben wird. Das Verlassen der darüber definierten inneren und

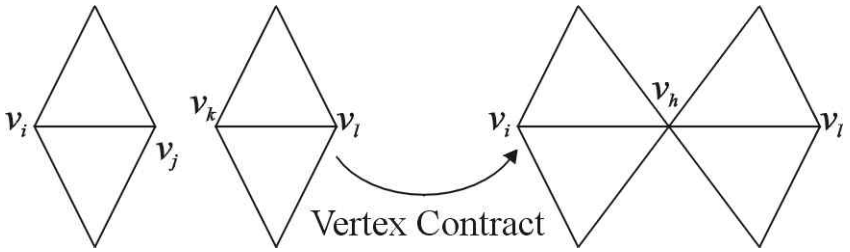

Abbildung 4.8: *Der Vertex Contract Operator kann die globale Topologie eines Polygonnetzes verändern.*

äußeren Hülle des Modells durch die transformierte Geometrie ist gleichbedeutend mit dem Abbruch der Iteration. Somit sind die Fehlermetriken und die mit ihnen einhergehenden Fehlerschranken wichtige Kriterien für das visuelle Ergebnis der Simplifizierung. Ein anderes Kriterium ist die Vorgabe der maximalen Polygonzahl. In diesem Fall werden solange Transformationen durchgeführt bis das resultierende Mesh die vorgeschriebene Polygonzahl unterschreitet. Zwar kann hiermit eine bessere Anpassung an die Leistungsmerkmale eines Endgerätes erfolgen, jedoch ist das Ergebnis nur schwer qualitativ abzuschätzen.

Die Auswahl einer bestimmten Fehlermetrik hat möglicherweise weitreichende Konsequenzen, die über das bloße Qualitätsmaß hinausgehen. So erfordert eine optimale Transformation mit minimalen Fehler einen immensen Rechenaufwand. Zudem werden topologische Informationen des Meshes benötigt, so dass sich eine Metrik bis auf die zu Beginn dieses Kapitels beschriebenen Datenstrukturen auswirken kann. Eine aufwändige Datenstruktur erfordert wiederum einen massiven Speicherverbrauch mit entsprechender Verwaltung. Ein weiterer Aspekt sind die bereits angesprochenen Zusatzattribute, die von einer Metrik gegebenenfalls berücksichtigt werden müssen.

In den letzten Jahren wurden viele Algorithmen zur Simplifizierung von Dreiecksnetzen entwickelt. Entsprechend den verwendeten Operatoren lassen sie sich in Verfahren klassifizieren, die entweder die globale Topologie erhalten oder verändern. Viele Vorgehensweisen können jedoch so modifiziert werden, dass sie im Gegensatz zum ursprünglichen Algorithmus die globale Topologie beeinflussen beziehungsweise nicht beeinflussen. Aus diesem Grund sind die Übergänge zwischen den beiden Klassen nicht eindeutig definierbar. Ein Beispiel hierfür ist das topologieverändernde Progressive Simplicial Complexes Verfahren [PH97], welches auf dem topologieerhaltenden *Progressive Meshes* basiert [Hop96].

Hoppes Prinzip der Progressive Meshes war aus dem Bedürfnis heraus entstanden, ein Element zuerst in einer sehr groben Struktur, dem Mesh M^0, an ein entferntes Endgerät zu übertragen und im weiteren Verlauf durch den schrittweisen Transfer kleiner Datenblöcke nach und nach zu verfeinern. Der Algorithmus von Hoppe ist nicht nur der Uralgorithmus in der progressiven Datenübertragung, Hoppe ist auch der Schöpfer dieser Terminologie und hat den Begriff *progressiv* für die Verwaltung seiner Meshes eingeführt und geprägt. Das Verfahren generiert aus einem Mesh $\hat{M}$, das weder non-manifold ist noch über doppelte Kanten oder Knoten verfügt, das vereinfachte Mesh M^0 und zusätzlich eine Sequenz von Informationen, über die aus M^0 inkrementell wieder $\hat{M}$ rekonstruiert werden kann. Für die Vereinfachung wird als atomare Operation der Edge Collapse Operator verwendet. Bei der Rekonstruktion wird die Umkehroperation, der Vertex Split Operator, an den Stellen eingesetzt, die zuvor

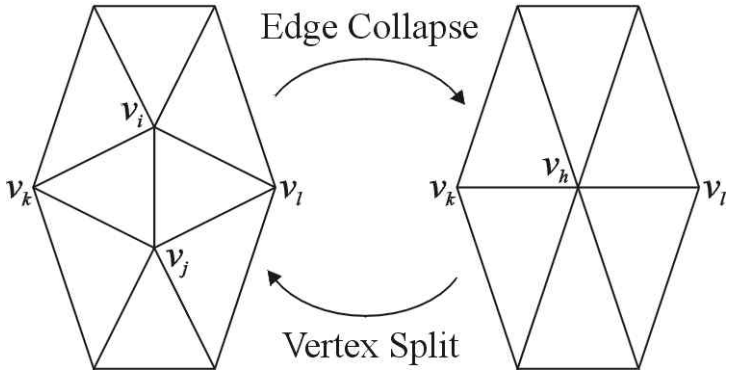

Abbildung 4.9: *Der Vertex Split Operator ist die Umkehroperation zur Kantenkollabierung.*

in den Zusatzinformationen der vereinfachten Meshes M^i mit $0 \geq i \geq n$ abgespeichert wurden. Ein Mesh M^n kann somit über ein simplifiziertes Mesh M^0 und eine Folge von Vertex Split Operationen $vsplit_k$ mit $0 \geq k < n$ beschrieben werden. Das Basismesh M^0 und die Transformationen ergeben zusammen ein *Progressive Mesh*.

$$M^0 \overset{vsplit_0}{\longrightarrow} M^1 \overset{vsplit_1}{\longrightarrow} \ldots \overset{vsplit_{n-1}}{\longrightarrow} (M^n = \hat{M})$$

Hoppe definiert ein Mesh über ein Tupel $M = (K, V)$. K ist ein *Simplicial Complex*, der topologische Beziehungen zwischen Scheitelpunkten, Kanten und Polygonen beinhaltet [HDDM93]. Ein Simplicial Complex K besteht aus einer Menge von Scheitelpunkten $V = \{v_1, \ldots, v_m\}$ und einer Menge von Teilmengen dieser Scheitelpunkte, *Simplices* genannt, so dass jede Menge mit genau einem Scheitelpunkt ein Simplex in K ist und jede nichtleere Teilmenge eines Simplex in K wieder einem Simplex in K entspricht [Spa66]. Ein d-dimensionales Simplex ist dabei ein d-dimensionales Polytop mit $d + 1$ Punkten, also die konvexe Hülle von $d + 1$ Punkten [GO97]. Entsprechend repräsentieren die 0-Simplizes $\{i\} \in K$ Scheitelpunkte, die 1-Simplizes $\{i, j\} \in K$ Kanten und die 2-Simplizes $\{i, j, k\} \in K$ Polygone beziehungsweise Dreiecke.

Um nun die geometrische Realisierung eines Meshes als Oberfläche im R^3 zu erhalten, muss zuerst für einen gegebenen Simplicial Complex K die topologische Realisierung $|K|$ im R^m bestimmt werden. Dies erfolgt über die Identifizierung der Scheitelpunkte $\{v_1, \ldots, v_m\}$ als Basisvektoren $\{e_1, \ldots, e_m\}$ des R^m. Für jedes Simplex $s \in K$ sei nun $|s|$ die konvexe Hülle seiner Scheitelpunkte im R^m und $|K| = \bigcup_{s \in K} |s|$. Sei weiterhin $\phi : R^m \mapsto R^3$ die lineare Abbildung, welche jedem $e_i \in R^m$ den entsprechenden Scheitelpunkt $v_i \in R^3$ zuordnet. Dann entspricht die geometrische Realisierung eines Meshes M dem Bild $\phi_V(|K|)$. Der Bezeichner ϕ_V für die Abbildung soll verdeutlichen, dass selbige durch die Menge der Scheitelpunkte $V = \{v_1, \ldots, v_m\}$ vollständig spezifiziert ist. Ist das Mesh und somit $\phi_V(|K|)$ nicht selbstüberschneidend, dann handelt es sich bei ϕ_V um eine $1 : 1$ Abbildung. In diesem Fall kann jeder Punkt $p \in \phi_V(|K|)$ durch das Finden seines eindeutigen Urbildes in $|K|$ parametrisiert werden. Der Vektor $b \in |K|$ mit $p = \phi_V(b)$ entspricht dem *Baryzentrischem Vektor* von p.

Grundsätzliches Ziel ist es, ein Mesh $M = (K, V)$ zu bestimmen, welches eine Menge X von Punkten $x_i \in R^3$ gut approximiert und die kleinstmögliche Anzahl an Knoten enthält. Hoppe löst dieses Problem durch das Minimieren einer Energiefunktion.

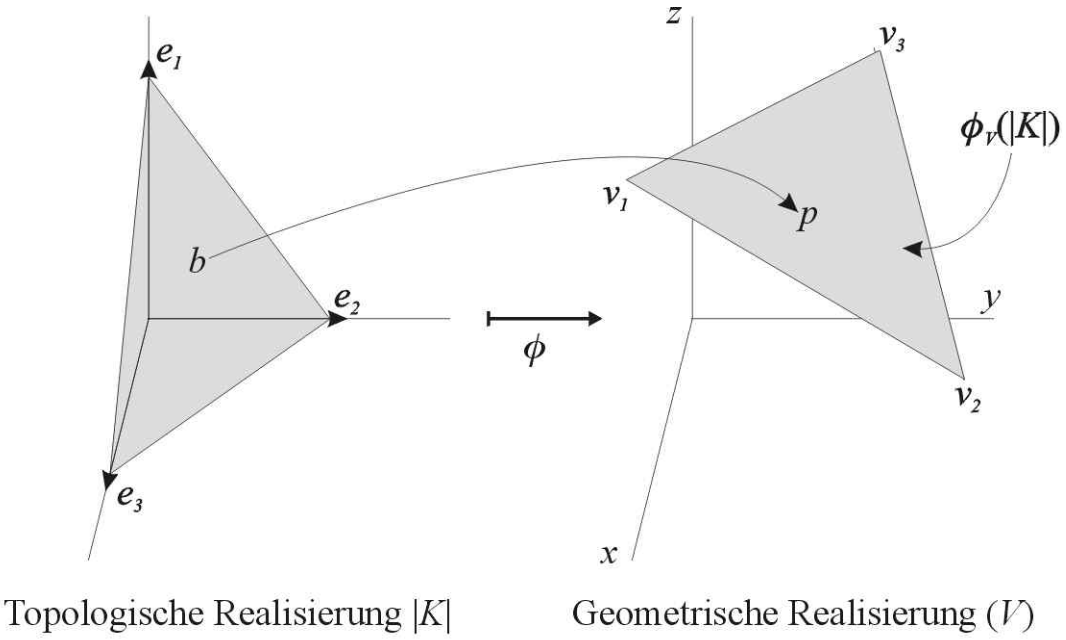

Abbildung 4.10: *Die geometrische Realisierung eines Meshes entspricht der Abbildung der topologischen Realisierung in R^3.*

$$E(M) = E_{dist}(M) + E_{spring}(M) + E_{scalar}(M) + E_{disc}(M)$$

$E_{dist}(M)$ steht hier für die Distanzenergie.

$$E_{dist}(M) = \sum_i d^2(x_i, \phi_V(|K|))$$

Die Distanzenergie $E_{dist}(M)$ sorgt dafür, dass sich das vereinfachte Mesh $M = (K, V)$ nicht zu weit vom ursprünglichen Mesh $\hat{M}$ entfernt. Dazu werden die Abstände der Punkte x_i zu M gemessen und deren Quadrate aufsummiert. Die Punkte x_i sind das Resultat einer vorangehenden Abtastung des Originalmeshes $\hat{M}$. Im Minimalfall wird jeder Scheitelpunkt von $\hat{M}$ abgetastet, es können aber bei Bedarf weitere hinzugenommen werden. Die Berechnung jedes einzelnen Abstandes ist für sich ein Minimierungsproblem, wodurch wiederum $E_{dist}(M)$ selbst minimiert wird.

$$d^2(x_i, \phi_V(|K|)) = \min_{b_i \in |K|} \|x_i - \phi_V(b_i)\|^2$$

Die topologische Realisierung von K im R^m, hier mit $|K|$ bezeichnet, wird durch die Definition der Knoten $\{1, \ldots, m\}$ als Basisvektoren $\{e_1, \ldots, e_m\}$ des R^m konstruiert. Für jedes Simplex $s \in K$ sei nun $|s|$ seine konvexe Hülle im R^m. Weiterhin sei $|K| = \bigcup_{s \in K} |s|$. Dann ist $\phi : R^m \mapsto R^3$ eine lineare Abbildung, welche den i-ten Basisvektor $e_i \in R^m$ nach $v_i \in R^3$ abbildet.

In dieser Gleichung ist die unbekannte Größe der baryzentrische Vektor $b_i \in |K| \subset R^m$ der Projektion von x_i auf das Mesh M. Bei gegebenen Punkten entspricht dies der Projektion

von x_i auf das Modell $\hat{M}$. Wenn nun die Kollabierung einer konkreten Kante v_s, v_t auf einen Scheitelpunkt v_s^i ansteht, so wird dessen Position wie folgt bestimmt:

$$v_s^i = (1-\alpha)v_s^{i+1} + \alpha v_t^{i+1}$$

Dabei entstammt α dem Wertebereich $\left\{0, \frac{1}{2}, 1\right\}$, d.h. die resultierenden Punkte entsprechen v_s, v_t oder dem arithmetischen Mittel zwischen v_s und v_t. Letztlich wird die beste dieser drei Möglichkeiten ausgewählt. Prinzipiell ist auch eine wirklich optimale Berechnung des Punktes möglich, was aber einem enormen Rechenaufwand bedarf.

Da eine alleinige Berücksichtigung der Distanzenergie für ein gutes visuelles Ergebnis der Vereinfachung nicht ausreichend ist, wird für jede Kante im Mesh M eine Feder mit der Federkonstanten κ angenommen.

$$E_{spring}(M) = \sum_{\{j,k\} \in K} \kappa \left\| v_j - v_k \right\|^2$$

Die Minimierung der Energie $E_{spring}(M)$ sorgt für eine optimale Verteilung der Dreiecke über das Mesh. Hierdurch beinhaltet ein simplifiziertes Modell keine unregelmäßig verteilten Dreiecke beziehungsweise Dreiecke, die sich vom Flächeninhalt extrem unterscheiden. In praktischen Anwendungen wird κ nicht global bestimmt, sondern bei jeder Kantenkollabierung. Dabei wird das Verhältnis zwischen der Anzahl der Punkte zu der Anzahl der Dreiecke in der Nachbarschaft der zu reduzierenden Kante gebildet. Je größer diese Zahl ist, umso kleiner fällt κ aus

Hoppe klassifiziert mögliche Zusatzattribute eines Meshes in skalare und diskrete Merkmale und erweitert die Beschreibung eines Mesh zum Tupel $M = (K, V, D, A)$. Die skalaren Attribute wie etwa Normalen oder Texturkoordinaten bilden die Menge A und werden den Scheitelpunkten des Meshes zugeordnet. Die Menge der diskreten Attribute D beschreibt dagegen die Eigenschaften der Polygone. Entsprechend wird ein Merkmal dieser Menge mit einem Polygon assoziiert.

Die Energie $E_{scalar}(M)$ misst die Abweichung der skalaren Attribute.

$$E_{scalar}(M) = (c_{scalar})^2 \sum_i \left\| x_i - \phi_v(b_i) \right\|$$

Der Wert c_{scalar} repräsentiert einen Gewichtungsfaktor und b_i eine Parametrisierung für das jeweilige Attribut.

Die Energie $E_{disc}(M)$ beschreibt die Oberflächenveränderung und zeichnet sich dafür verantwortlich, dass ein simplifiziertes Modell in Bereichen mit starker Krümmung nur geringfügig vom Original abweicht. Entsprechende Regionen prägen oftmals die Charakteristik eines Modells, weshalb hier große Veränderungen die Qualität eines Modells sehr beeinträchtigen.

Der Ansatz von Popovic und Hoppe [PH97] ist dem Verfahren von Hoppe sehr ähnlich. Einer der Unterschiede besteht in der Einführung einer neuen Transformation für die Vereinfachung eines Modells. Popovic et al. nennen diese Transformation *generalized vertex split*,

also eine Verallgemeinerung des Vertex Splits beziehungsweise des Knotenteilens. Eine weitere Innovation beruht auf der Betrachtungsweise der elementaren Bausteine eines Modells. Hier werden nämlich nicht die Punkte, sondern die Simplices eines Modells berücksichtigt.

Im Jahr 1997 stellte Schröder eine progressive Erweiterung [Sch97] seines 1992 mit Zarge und Lorensen veröffentlichten Ansatzes [SZL92] vor. Da das Verfahren die Topologie eines Modells verändert, erreicht es eine hohe Datenreduktion. Die elementare Operation ist die Kantenkollabierung, wobei die Einhaltung der Fehlerschranken zwar garantiert werden kann, aber nicht gewährleistet ist, dass bei vorgegebener Fehlerschranke die minimale Anzahl an Dreiecken erreicht wird. Eine Neuerung des Ansatzes liegt in der Klassifizierung der Punkte eines Modells noch vor seiner eigentlichen Vereinfachung. Hierdurch werden beispielsweise degenerierte Dreiecke, Randkanten und Risse eines Modells identifiziert. Die Intention von Schröder basiert auf der vollautomatischen Konvertierung von Daten aus einer Modelldatenbank, weshalb eine entsprechende Klassifizierung eine hohe Robustheit des Verfahrens ermöglicht.

Ein weiterer bekannter Ansatz zur Geometriereduktion stammt von Garland und Heckbert [GH97]. Sie verwenden eine *Quadric Error Metric* zur Bestimmung des Fehlers eines Scheitelpunkts. Während in vielen anderen Verfahren die Kantenkollabierung zum Einsatz kommt, basiert die Vorgehensweise von Garland et al. auf der Kontraktion zweier Punkte. Ein Knotenpaar (v_1, v_2) resultiert in einem einzelnen Knoten v, indem v_1 und v_2 an eine neue Position verschoben und alle Verbindungen zu v_2 an v_1 gehängt werden. Anschließend wird v_2 gelöscht. Eine Besonderheit der Kontraktion beruht darauf, dass zwischen v_1 und v_2 keine gemeinsame Kante existieren muss. Im Jahr 1998 stellten Garland und Heckbert eine Variante ihres Algorithmus vor, welche Texturkoordinaten berücksichtigt. Allerdings sind beide Vorgehensweisen nicht progressiv.

In den meisten Fällen ist das Entfernen eines Punktes aus einem Polygonnetz mit einem Verlust an Informationen und somit auch an Detail verbunden. Einige Verfahren versuchen dies zu vermeiden, indem sie das durch den entfernten Punkt entstehende Loch retriangulieren. Äquivalent zu den anderen Ansätzen werden die entsprechenden Punkte ebenfalls über eine Metrik bestimmt. Ein Vorschlag für ein derartiges Verfahren stammt von Klein et al. [KLS96]. Sie legen großen Wert darauf, dass die Metrik den wirklichen Fehler misst. Somit ist es auch noch nach mehreren Iterationen möglich, den genauen Abstand des aktuell simplifizierten Modells zum Original zu berechnen. Andere Verfahren verwenden hierzu oftmals lediglich eine Approximation. Als Metriken kommen die verschiedenen 3D Versionen der *Hausdorff Distanz* zum Einsatz. Seien X und Y zwei Punktmengen, so ist die Hausdorff Distanz d_H und die einseitige Hausdorff Distanz d_E wie folgt definiert:

$$\begin{aligned} d_H(H,Y) &= max(d_E(X,Y), d_E(Y,X)) \\ d_E(X,Y) &= \sup_{x \in X} d(x,Y) \\ d(x,Y) &= \inf_{y \in Y} d(x,y) \end{aligned}$$

Mit $d(x,Y)$ wird dabei die *Euklidische Distanz* eines Punktes zu einer Punktmenge bezeichnet.

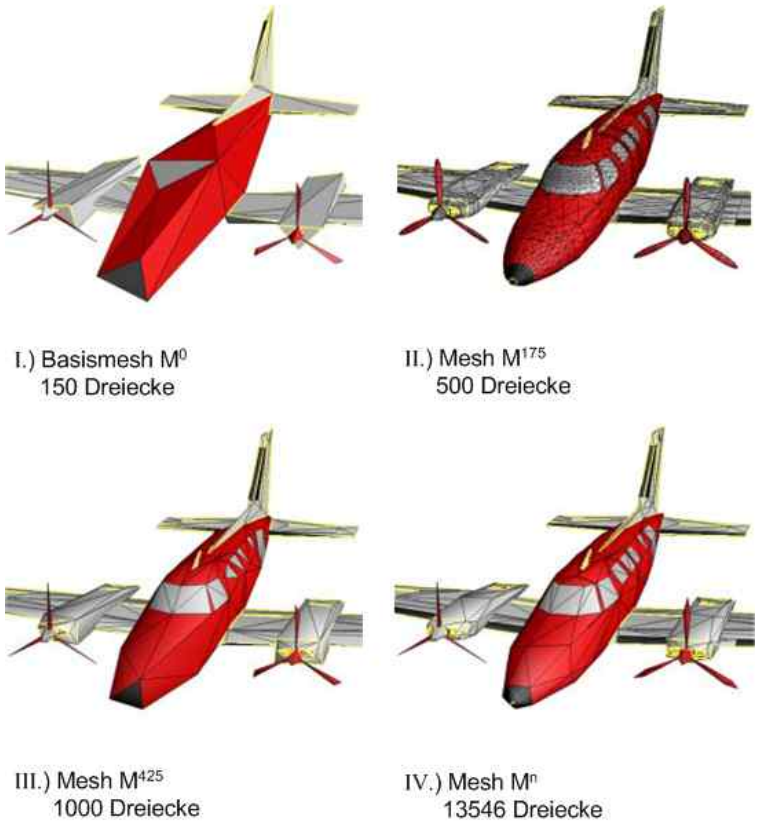

Abbildung 4.11: *Die simplifizierte Chessna von Hoppe in verschiedenen Detailstufen.*

Klein et al. schlagen aus Performanzgründen die Verwendung der einseitigen Hausdorff Distanz d_E vor, da die symmetrische Hausdorff Distanz d_H nur in ganz speziellen Fällen eingesetzt werden muss. Diese beschränken sich auf Randkanten und Kanten, an denen die Winkel der angrenzenden Dreiecke sehr spitz sind. Klein et al. erreichen unter Vorgabe der Fehlerschranke eine höhere Datenreduktion als andere Verfahren. Allerdings ist ihr Ansatz ebenfalls wie der von Garland et al. nicht progressiv.

4.3.2 Resampling

Eine weitere Gruppe von Verfahren zur automatischen Generierung von Level-of-Detail wurde 1993 von Jarek Rossignac und Paul Borrel eingeführt [RB93]. Ihr Algorithmus ist einer der wenigen, die weder eine gültige Topologie erfordern noch sie in irgendeiner Weise erhalten. Hierdurch bietet der Ansatz den Vorteil, auch degenerierte Modelle verarbeiten zu können, was insbesondere bei von Hand erstellten CAD Datenbanken von Bedeutung ist.

Im ersten Schritt des Verfahrens wird jedem Scheitelpunkt eines Modells eine Priorität in Bezug auf das äußere Erscheinungsbild des Modells zugewiesen. Diese Priorisierung basiert auf zwei Faktoren: Zum einen werden von größeren Dreiecken indizierte Scheitelpunkte höher priorisiert als von kleineren Dreiecken indizierte Scheitelpunkte. Zum anderen erhalten

Scheitelpunkte innerhalb stärker gekrümmten Bereichen eine höhere Priorität als Scheitelpunkte in nahezu planaren Umgebungen. Im zweiten Schritt wird die Bounding Box des Modells bestimmt und in ein reguläres Gitter unterteilt. Der Algorithmus reduziert alle Punkte innerhalb einer Zelle des Gitters auf einen einzelnen repräsentativen Scheitelpunkt, welcher anhand der zuvor ermittelten Priorität ausgewählt wird. Die Auflösung des Gitters bestimmt das Maß der Simplifizierung: Während ein grobes Gitter eine sehr starke Reduktion verursacht, impliziert ein entsprechend feines Gitter lediglich geringe Änderungen.

Neben seiner Robustheit ist der Ansatz von Rossignac und Borrel einfach zu implementieren und verfügt über eine geringe Laufzeit. Allerdings hat das Verfahren auch signifikante Nachteile: Da weder die Topologie erhalten noch eine Fehlerschranke in Bezug auf die Oberfläche des Modells eingehalten wird, ist das visuelle Ergebnis oftmals schlechter als von langsameren Algorithmen. Zudem ist das resultierende Level-of-Detail von der Orientierung des Gitters abhängig, weshalb die Simplifizierung des gleichen Modells mit unterschiedlichen Orientierungen auch differierende Ergebnisse produziert. Ein weiterer Nachteil ist, dass die Anzahl der verbleibenden Dreiecke nicht vorgegeben oder abgeschätzt werden kann. Hier besteht lediglich die Möglichkeit, die Anzahl über das Rendering des Modells zu bestimmen.

Aufgrund dieser Nachteile führten Kok-Lim Low und Tiow-Seng Tan im Jahr 1997 eine verbesserte Vorgehensweise ein [LT97]. Basierend auf der Feststellung, dass es sich bei dem Ansatz von Rossignac und Borrel letztlich um eine Form des Vertex Clustering [3] handelt, bezeichnen sie ihr Verfahren als Floating-Cell-Clustering. Ähnlich wie Rossignac und Borrel priorisieren Low und Tan die Scheitelpunkte eines Modells, verwenden dabei allerdings eine verbesserte Metrik. Um den Scheitelpunkt mit der höchsten Priorität legen sie eine Zelle von benutzerdefinierter Größe und kollabieren alle Scheitelpunkte innerhalb dieser Zelle auf den repräsentativen Vertex. Der verbleibende Scheitelpunkt mit der höchsten Priorität wird zum Mittelpunkt der nächsten Zelle und der Vorgang wiederholt sich. Die Verwendung des Floating-Cell-Clustering reduziert in starkem Maße den Einfluss von Position und Orientierung des Modells auf das Ergebnis der Simplifizierung. Zudem fällt die Variation der Resultate in Abhängigkeit von der gewählten Zellgröße wesentlich geringer aus als bei einem regulären Gitter. Aufgrund der Vernachlässigung einer Fehlerschranke im Ansatz von Rossignac und Borrel weisen Low und Tan auf den Bezug zwischen der Zellengröße und der maximalen Anzahl von Pixel hin, die von einer Zelle während der Rasterisierung belegt werden können. Hierdurch kann der Benutzer die Vorgabe treffen, dass ein Level-of-Detail erst dann verwendet wird, wenn die verbleibenden Scheitelpunkte innerhalb von n Pixel zueinander liegen. Basierend auf dieser Beobachtung entstand 1997 ein weiteres Verfahren, welches als eines der ersten eine interaktive Simplifizierung unter Berücksichtigung der Sichtpyramide eines Betrachters berücksichtigte [LE97].

4.3.3 Unterteilungsflächen

Unterteilungsflächen beziehungsweise *Subdivision Surfaces* erzeugen ausgehend von einem groben Basismesh durch rekursive Verfeinerung ein Mesh mit einer höheren Auflösung. Ziel

[3] Alle Scheitelpunkte innerhalb einer Zelle (Clusters) werden zu einem Scheitelpunkt kombiniert.

der Verfeinerung ist die Annäherung an das Originalmesh [4] mit Hilfe eines Regelsatzes. Das Ende der Rekursion ist erreicht, sobald das resultierende Mesh entweder eine Mindestzahl an Polygonen überschreitet oder sich vom Originalmesh nur noch durch einen vorgegebenen Schwellwert unterscheidet. Die grundlegende Idee der Unterteilungsflächen wurde ursprünglich von Chaikin vorgestellt [Cha74]. Seitdem hat es viele Arbeiten in diesem Bereich gegeben, die sich in approximierende und interpolierende Verfahren unterteilen lassen. Zwei wichtige approximative Ansätze stammen von Catmull et al. [CC78] und Loop [Loo87]. Die vielleicht bedeutendste interpolierende Vorgehensweise ist das *Butterfly Verfahren* [DLG90]. Eine Analyse spezifischer Schemata findet sich unter anderem in Kobbelt et al. [KS97].

Theoretisch benötigen Unterteilungsflächen nur einen geringen Speicheraufwand, da lediglich das Basismesh und das aktuelle Mesh gespeichert werden müssen, um Visualisierungen in nahezu beliebig verschiedenen Auflösung zu erhalten. Dem steht jedoch der immense Rechenaufwand gegenüber, der vor allem für hoch aufgelöste Meshes mit entsprechender Rekursionstiefe erforderlich ist. Selbst mit den Leistungsmerkmalen heutiger High-End-Workstations wäre es unmöglich, innerhalb jedes Frames die verschiedenen Meshes der sichtbaren Elemente einer großen Szene in Echtzeit zu generieren. Aus diesem Grund bieten sich Unterteilungsflächen als ein Verfahren zur Vorberechnung gewünschter Auflösungen an, wodurch sie zwar einen Beitrag zur Adaptivität der Daten leisten, aber letztlich keinen Speichervorteil offerieren können.

Ein weiteres Problem der Unterteilungsflächen betrifft den Modellierungsprozess. Wie kann durch Verfeinerung des Basismeshes das gewünschte Ergebnis erreicht werden? Wie wird das Basismesh selbst erzeugt? Dabei ist zu beachten, dass Basismesh und Limitmesh die gleiche Topologie aufweisen müssen, da die Unterteilung keine topologischen Veränderungen vornimmt. Aufgrund dieser Problematik ist die Generierung eines Meshes mit einer bestimmten Auflösung oftmals ein interaktiver Prozess, in dem der Anwender nach einer sorgfältigen Auswahl des Basismeshes innerhalb der einzelnen Rekursionsstufen Nachbesserungen vornehmen muss. Allerdings existieren mittlerweile Ansätze, die Werkzeuge für die Modellierung des Originalmeshes anbieten und den Anwender nicht darüber hinaus belasten [Qui00].

4.3.4 Multiresolution Analysis

Multiresolution Analysis wurde 1989 von Mallat [Mal89] eingeführt und für Funktionen im R^n definiert. Die grundlegende Idee beruht darauf, eine Funktion derart zu teilen, dass ein niederfrequenter Rest verbleibt und ein Teil an Informationen abgespaltet wird, welcher die Differenz des Rests zur Ausgangsfunktion darstellt. Hier sind bereits Parallelen zur progressiven Vorgehensweise erkennbar.

Im Jahr 1994 stellte Michael Lounsbery einen Ansatz vor [Lou94], um 3D Modelle zu repräsentieren. Das Verfahren bietet eine qualitativ hochwertige Simplifizierung sowie eine nahezu

[4] Das Originalmesh wird in diesem Zusammenhang auch als *Limit Mesh* bezeichnet. Mit dem Ansatz der Unterteilungsflächen ist es sogar möglich, eine feinere Auflösung als das Original zu bieten. Ob dies allerdings auch der Intention des Designers entspricht ist eine andere Frage.

stufenlose Auflösung der vereinfachten Modelle. Außerdem kann es für eine progressive Übertragung der Modelle verwendet werden.

Die elementaren Inhalte der Multiresolution Analysis entsprechen einer Menge von ineinander geschachtelten Funktionsräumen $V^0 \subset V^1 \subset \ldots$ und dem dazugehörenden inneren Produkt. Letzteres wird für die Definition der Orthogonalität der Funktionen verwendet. Der Funktionsraum V^j wird von einer Menge an Funktionen ϕ aufgespannt, wobei ϕ in diesem Zusammenhang auch als Skalierfunktion bezeichnet wird.

$$\begin{aligned} \phi(x) &: R \mapsto R \\ \phi(x) &= \sum_i p_i \phi(2x - i) \end{aligned}$$

Damit ist

$$V^j = span(\phi(2^j x - i); i \in Z)$$

Mit wachsendem j nimmt der Detailgrad und somit auch die Genauigkeit der Repräsentation zu. Bezüglich eines inneren Produkts kann nun ein Funktionsraum $V_\perp^j$ als ein orthogonales Komplement von V^j in V^{j+1} definiert werden:

$$V^{j+1} = V^j \oplus V_\perp^j$$

Nun kann die i-te Skalierfunktion bei einer Auflösung j wie folgt beschrieben werden:

$$\phi(x)_i^j = 2^{\frac{j}{2}} \phi(2^j x - i)$$

Der Faktor $2^{\frac{j}{2}}$ dient der Normierung. Wavelets sind nun Funktionen

$$\psi_i^j = 2^{\frac{j}{2}} \psi(2^j x - i),$$

die eine Basis für $V_\perp^j$ bilden. Gegeben sei eine Funktion $f^j(x)$ in V^j

$$f^j(x) = \sum_i c_i^j \phi_i^j(x)$$

Diese Funktion kann als eine Summe von zwei Komponenten geschrieben werden, die in V^{j-1} beziehungsweise in $V_\perp^j$ liegen.

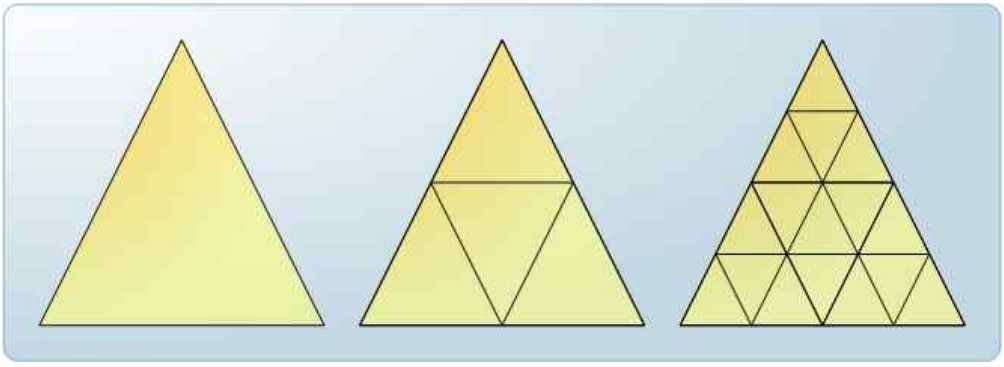

Abbildung 4.12: *Der Algorithmus von Lounsbery unterstützt lediglich ein 4:1 Splitting.*

$$f^j(x) = \sum_i c_i^{j-1}\phi_i^{j-1}(x) + \sum_i d_i^{j-1}\psi_i^{j-1}(x)$$

Damit existieren auch zwei Folgen $a = \ldots, a_{-1}, a_0, a_1, \ldots$ und $b = \ldots, b_{-1}, b_0, b_1, \ldots$, so dass

$$\begin{aligned} c^{j-1} &= \sum_l a_{l-2i} c_i^j \\ d_i^{j-1} &= \sum_l b_{l-2i} c_i^j \end{aligned}$$

Somit kann die ursprüngliche Folge c_i^j aus c_i^{j-1} und d_i^{j-1} rekonstruiert werden, sofern zwei weitere Folgen p und q zum Einsatz kommen.

$$c_i^j = \sum_i (p_{i-2l} c^{j-1} + q_{i-2l} d_l^{j-1})$$

Die Folgen a und b werden *Dekompositionsfilter* beziehungsweise *Analysefilter* und die Folgen p und q *Rekonstruktionsfilter* beziehungsweise *Synthesefilter* genannt. Das Ziel ist, c_i^j in ein c_i^{j-1} und d_i^{j-1} aufzuteilen. Während ersteres den gröberen Teil des Modells repräsentiert, beschreibt letzteres die verloren gegangenen Details. Dieser Vorgang kann rekursiv fortgesetzt werden. Er bedeutet die Dekomposition einer Funktion $f^i(x)$ in eine einfachere Funktion $f^0(x) \in V^0$ sowie eine Menge von Wavelet-Koeffizienten, welche benötigt werden, um die ursprüngliche Funktion zu rekonstruieren.

Der Algorithmus von Lounsbery hat einen gravierenden Nachteil: Er kann lediglich Modelle bearbeiten, die aus einem Modell durch 4-zu-1-Splitting gewonnen wurden. Abbildung 4.12 verdeutlicht dies. Eine Lösung hierfür bietet der Ansatz von Eck et al. [EDD+95].

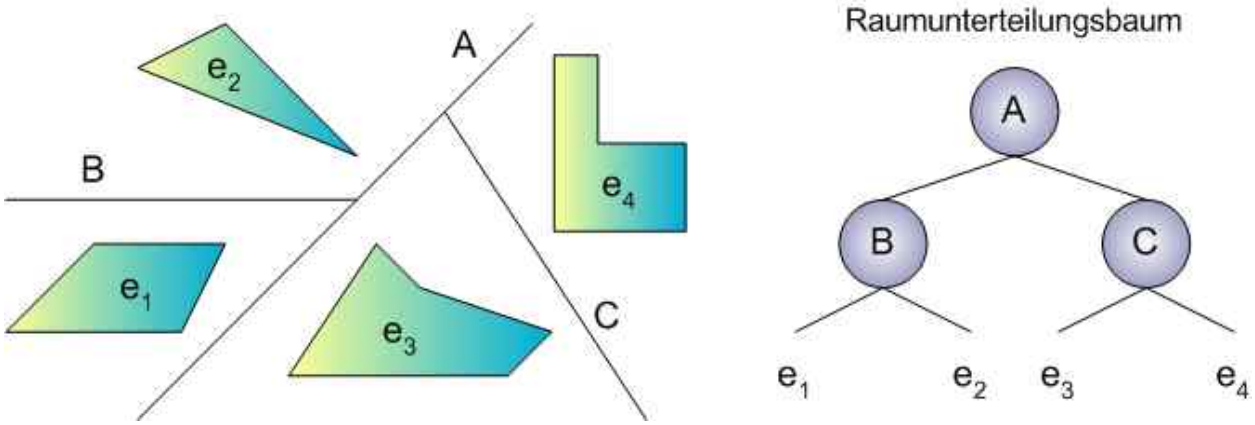

Abbildung 4.13: *In einem klassischen BSP Baum erfolgt die Unterteilung nicht nach achsenparallelen Hyperebenen, sondern orientiert sich an den Elementen der Szene. Entsprechend resultiert diese Vorgehensweise zumeist in balancierten Bäumen.*

4.4 Raumunterteilungsbäume

Raumunterteilungbäume oder auch *Space Partitioning Trees* werden in einer Vielzahl verschiedener Anwendungen eingesetzt, beispielsweise zur Lösung des Problems verdeckter Flächen [DBHOS94, Jam99, Mur99] (siehe auch Abschnitt 4.6), zur Kollisionserkennung [Nay92], zur Lichtstrahlverfolgung [CSM95, ACT00] oder zur Modellierung [Hun78, RR78, JT80, Mea80]. Innerhalb dieser Arbeit dienen sie ausschließlich zur räumlichen Sortierung der Elemente einer 3D Szene. Ziel ist dabei die hierarchische Repräsentation von räumlichen Kohärenzen zwischen diesen Elementen: Mehrere benachbarte Elemente werden in einer Menge vereinigt, welche wiederum mit anderen benachbarten Mengen zu einer übergeordneten Menge vereinigt wird. Mit Hilfe dieser hierarchischen Darstellung lassen sich verallgemeinerte Aussagen treffen, die dann nicht nur für ein einzelnes Element, sondern für eine gesamte Menge von Elementen gültig sind. Wird beispielsweise die gemeinsame Bounding Box aller Elemente einer Menge bestimmt und diese in Bezug auf einen Betrachter als unsichtbar erkannt, so sind auch alle Elemente innerhalb der Menge unsichtbar. Im Falle der Kollisionserkennung wäre eine Annahme möglich, wonach die Elemente zweier räumlich disjunkter Mengen nicht mit Elementen der jeweils anderen Menge kollidieren können. Ein entscheidendes Problem ist die Pflege, d.h. die konsistente und effiziente Aufrechterhaltung der hierarchischen Repräsentation innerhalb dynamischer Szenen, da hier die Nachbarschaftsbeziehungen der Elemente einer permanenten Änderung unterliegen.

Ursprünglich wurde das Prinzip der Raumunterteilungsbäume 1969 von Schumacker et al. [SBGS69] als Lösung des Problems verdeckter Flächen eingeführt, indem sie mit Hilfe eines *Binary Space Partitioning* (BSP) Baumes die Polygone einer Szene in Bezug auf einen Betrachter in einer *back-to-front* Reihenfolge sortieren. Jedes Polygon der Szene repräsentiert dabei eine Hyperebene h, welche allgemein formuliert im d-dimensionalen Raum R^d mit Hilfe einer impliziten Funktion $H(x_1, \ldots, x_d) = a_{d+1} + \sum_{i=1}^{d} a_i x_i = 0$ dargestellt werden kann [5]. Sei $h^+ = \{(x_1, \ldots, x_d) | H(x_1, \ldots, x_n) > 0\}$ der von h begrenzte positive offene Halbraum

[5] Da in diesem Fall die Polygone selbst die Hyperebenen repräsentieren, wird auch von einer *Autopartition* gesprochen.

und $h^- = \{(x_1, \ldots, x_d) | H(x_1, \ldots, x_n) < 0\}$ der negative offene Halbraum. Sei weiterhin die Konstante δ die Kapazität eines Baumes T. Dann gelten für einen BSP Baum, welcher eine Szene S mit n Elementen $E_1, \ldots, E_n$ repräsentiert, die folgenden beiden Regeln:

- Ist die Anzahl der Elemente $|S| \leq \delta$, dann ist T ein einzelner Blattknoten, dem alle Elemente der Szene zugeordnet sind.
- Ist die Anzahl der Elemente $|S| > \delta$, dann muss der durch die Wurzel w von T repräsentierte Raum mittels einer Hyperebene h_w unterteilt werden, woraus sich die Halbräume H_w^+ und H_w^- ergeben. Die Information über h_w wird dabei in w gespeichert. Der linke Kindknoten der Wurzel w repräsentiert den Halbraum H_w^- und der rechte Kindknoten den Halbraum H_w^+. Alle Elemente, die vollständig in H_w^- liegen, werden dem linken Kindknoten zugeordnet, alle Elemente, die vollständig in H_w^+ liegen, dem rechten Kindknoten. Somit beinhaltet der linke Kindknoten die Teilmenge $S_w^- \subset S$ und der rechte Kindknoten die Teilmenge $S_w^+ \subset S$.

Übersteigt die Anzahl der Elemente innerhalb der Kindknoten die Kapazität δ, so muss dort der Vorgang rekursiv fortgesetzt werden.

Die letzte der oben aufgeführten Regeln verdeutlicht bereits einige signifikante Probleme der Raumunterteilungsbäume: Die erste Hürde ist die Wahl einer geeigneten Hyperebene h_w für die Unterteilung von w, welche optimalerweise in einem balancierten Baum resultieren sollte. Die Vorgehensweise bei der Bestimmung der Hyperebene h_w wird im folgenden auch als Strategie des Raumunterteilungsbaumes bezeichnet. Die zweite Hürde ist die Frage, was mit Elementen geschieht, die sich nicht eindeutig in einen der entstehenden Halbräume zuordnen lassen. Derartige Elemente werden im folgenden auch als Schnittelemente bezeichnet. Unter dem Gesichtspunkt, dass in dieser Arbeit dynamische Szenen berücksichtigt werden sollen, ist das erste Problem mit Hilfe eines generellen BSP Baumes derzeit für eine große Anzahl an Primitiven nicht in Echtzeit lösbar. Für das zweite Problem geben Greene et al. [GKM93] einige Lösungsvorschläge:

- Das betreffende Element wird an der Hyperebene geclippt und die resultierenden Fragmente den entsprechenden Halbräumen zugeordnet. Diese Vorgehensweise erfordert nicht nur einen hohen Rechenaufwand für das Clippen der Elemente, sondern auch einen hohen Verwaltungsaufwand. Wird beispielsweise von dem Konzept der Animationselemente (siehe Abschnitt 4.2) ausgegangen, dann müsste die geometrische Information des Animationselements unterteilt werden und gleichzeitig die Assoziation mit den anderen Informationen erhalten bleiben. Zudem steigt mit wachsender Zahl der Fragmente die Komplexität des Baumes.
- Das Element wird beiden Halbräumen zugeordnet. Auch hier entsteht bei dynamischen Szenen ein großer Verwaltungsaufwand, da in diesem Fall mehrere Knoten des Baumes von der Transformation eines einzelnen Elements betroffen sein können.
- Das Element bleibt in dem aktuellen Knoten. In diesem Fall würden zum Beispiel alle Elemente, welche h_w schneiden, dem Wurzelknoten w zugeordnet. Im Falle der von

Schumacker et al. adressierten Anwendung würde dies zu einer unvollständigen Sortierung führen. Das grundsätzliche Problem hierbei ist, dass die Elemente in einem sehr niedrigem Level des Baumes einsortiert werden, weshalb bei einer Traversierung des Baumes viele Elemente unnötigerweise bearbeitet werden müssen. Wird etwa nach einem bestimmten Element unter Vorgabe einer Position gesucht, dann ist zwar der entsprechende Knoten schnell gefunden, die dort einsortierten Elemente erfordern jedoch eine Überprüfung mit linearem Aufwand.

Um das zuvor erwähnte Problem der Bestimmung einer geeigneten Hyperebene h_v zur Unterteilung eines Knoten v (beziehungsweise des von v repräsentierten Raumes) einzuschränken, wurden weitere Typen von Raumunterteilungsbäumen entwickelt. Ein wichtiger Vertreter ist dabei der *k-D-Tree*, welcher von Bently [Ben75, Ben79] für die assoziative Suche innerhalb eines k-dimensionalen Suchraumes vorgestellt wurde. Letztlich ist der k-D-Tree eine Unterart des BSP Baumes, da er lediglich die Wahl der Hyperebenen auf achsenparallele Ebenen restriktiert, ansonsten aber auf dem gleichen Prinzip basiert. Im dreidimensionalen Fall gibt es somit drei verschiedene Richtungen, in die ein Knoten unterteilt werden kann, nämlich in x-, y- und z-Richtung. Im klassischen k-D-Tree Ansatz erfolgt die Unterteilung in einer fest vorgegebenen alternierenden Reihenfolge, etwa $x, y, z, x, \ldots$. Hierdurch wird die Suche nach einer geeigneten Hyperebene h_v zwar vereinfacht, allerdings kann die Effizienz des Baumes darunter leiden, weshalb neuere Einsätze derartige Vorgaben nicht mehr tätigen. Auch mit der Einschränkung achsenparalleler Hyperebenen ist die Suche geeigneter h_v ein zeitintensives Unterfangen. Aus diesem Grund nehmen einige Ansätze einfach eine Halbierung des durch v repräsentierten Raumes vor [COS94, Sam89], andere Ansätze eine willkürliche Unterteilung [APB87] und wiederum andere Ansätze eine Unterteilung, die sich nach dem Median der Elemente [6] richtet [DBVKOS97]. Eine weitere Gruppe von Verfahren sucht die Hyperebenen zwischen Raummedian und Elementmedian [MB90, Sub90, WSC+95]. In [BMH99] schlagen Bartz et al. in Form der *Sloppy-n-Array* eine Aufweichung des Unterteilungsprinzips vor, wobei die durch einen Knoten repräsentierten Räume nicht mehr vollständig disjunkt zueinander sind, sondern sich innerhalb eines Toleranzbereiches überlappen können.

Der *Octree* ist ebenfalls eine Variante der Raumunterteilungsbäume. Er besitzt ähnliche Eigenschaften wie der k-D-Tree, wobei allerdings eine Unterteilung zum einen immer im Raummedian und zum anderen in alle Dimensionen gleichzeitig erfolgt. Im dreidimensionalen Fall ergeben sich somit für einen Knoten v drei zueinander orthogonale Hyperebenen h_{vx}, h_{vy} und h_{vz}, deren Schnittpunkt das Zentrum des von v repräsentierten Raumes bilden. Für diesen durch einen Knoten v beschriebenen Raum gibt es eine Vielzahl verschiedener Bezeichnungen, unter anderem *Obel* [FA85], *Prism* [Gla88], *Voxel* [WSC+95], *Zelle* [RAJ96], *Würfel* [AF99] oder *Octree Box* [CD99]. Im folgenden wird unabhängig vom Typ des Raumunterteilungsbaumes der von Hyperebenen eingeschlossene Raum eines Knotens als Zelle bezeichnet. Da Zellen über kein visuelles Erscheinungsbild verfügen, wird zur Verdeutlichung auch von virtuellen Zellen gesprochen. Im traditionellen Octree verwalten lediglich die Blätter des Baumes die Elemente der Szene [Gla84], eine Speicherung in den inneren Knoten ist jedoch ebenfalls möglich [FI85]. Das zweidimensionale Äquivalent zum Octree stellt der Quadtree dar.

[6] Die Unterteilung erfolgt in der Mitte des von den Elementen aufgespannten Raumes.

Für die Terminierung der rekursiven Konstruktion gelten bei Octree und k-D-Tree die gleichen Kriterien:

- Die Anzahl der Elemente $|S_v|$ eines Knoten v ist kleiner als die Konstante δ.
- Die Rekursion erreicht eine vorgegebene maximale Tiefe.
- Die Dimensionen der aktuellen Zelle erreichen eine minimale Grenze.

Aufgrund der Unterteilung mit drei Hyperebenen fällt die Tiefe eines Octrees im Vergleich zu einem dreidimensionalen k-D-Tree deutlich geringer aus. Bei einer Suche nach einem bestimmten Element unter Vorgabe der Position wird somit der entsprechende Knoten des Baumes im Octree schneller gefunden als im k-D-Tree, jedoch kann sich die Suche im Knoten selbst aufwändiger gestalten.

4.5 Bounding Volume Hierarchien

Bounding Volume Hierarchien stellen eine Lösung zu dem in Abschnitt 4.4 beschriebenen Problem der Schnittelemente dar. Bei der Konstruktion einer Bounding Volume Hierarchie ist zwischen der bottom-up und der top-down Methode zu unterscheiden.

Der bottum-up Ansatz betrachtet jeweils n Elemente einer Szene und bestimmt über deren Geometrie ein die Elemente umfassendes Volumen. Je nach Ansatz werden unterschiedliche Formen der Volumina verwendet, beispielsweise Kugeln, achsenparallele Würfel oder Zylinder. Das resultierende Volumen einschließlich der entsprechenden Elemente wird einem Blattknoten der Hierarchie zugeordnet, von denen wiederum n Knoten zu einem Volumen zusammengefasst und durch einen gemeinsamen Vaterknoten repräsentiert werden. Dieser Vorgang wird fortgesetzt bis nur noch ein gemeinsamer Vaterknoten als Wurzel des balancierten Baumes übrig bleibt. Der Wert n spezifiziert die Anzahl der Kinder eines Knotens und wird daher auch als *Branching Factor* bezeichnet. Ein großes Problem bei der buttom-up

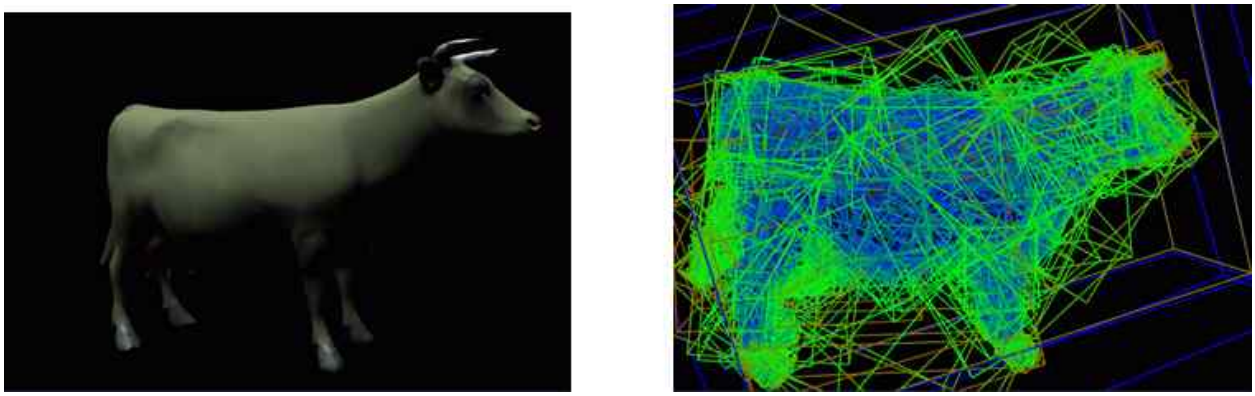

Abbildung 4.14: *Die räumliche Unterteilung eines Modells durch Bounding Volumes. In diesem Fall sind die Volumina nicht achsenparallel, sondern richten sich nach der Orientierung ihres Inhalts.*

Konstruktion von Bounding Volume Hierarchien ist die Identifikation von Nachbarschaftsbeziehungen zwischen den Elementen beziehungsweise der Volumina. Bei einer Vernachlässigung dieser Beziehungen können sehr große Zellen mit weitreichenden gegenseitigen Überlappungen entstehen. Derartige Überlappungen machen das Suchen innerhalb des Baumes unter Vorgabe einer Position und andere Applikationen wie etwa Sichtbarkeitsbestimmungen (siehe Abschnitt 4.6), Lichtstrahlverfolgungen oder Kollisionserkennungen äußerst ineffizient. Aus diesem Grund nehmen Weghorst et al. und Kay et al. [KK86, WHG84] zuerst eine Sortierung der Elemente entlang der x-Achse vor.

Der top-down Ansatz von Kay et al. [KK86] sortiert die Elemente der Szene ebenfalls entlang der x-Achse. Nach der Sortierung werden die Elemente in zwei von der Zahl her nahezu gleich große Mengen unterteilt (d.h. $n = 2$) und diese den Kinderknoten der Wurzel zugeordnet. Dort erfolgt wiederum eine Partitionierung der jeweiligen Teilmengen in zwei gleich große Mengen. Der Vorgang wird rekursiv fortgesetzt bis nur noch ein Element pro Teilbaum übrig ist. Das Ergebnis der Rekursion ist ein balancierter Baum. Im Gegensatz zu Kay et al. sortiert das Verfahren von Smits [Smi98] die Elemente nicht nur in x-Richtung, sondern nimmt in jedem Level eine Sortierung der Elemente in alternierender Richtung vor.

Der Vorteil von Bounding Volume Hierarchien sind die resultierenden balancierten Bäume, deren Blätter die Elemente der Szene verwalten. Somit sind die Elemente maximal tief eingeordnet, weshalb das in Abschnitt 4.4 erwähnte Problem entfällt, wonach in niedrigen Level einsortierte Elemente bei einer Traversierung unnötigerweise prozessiert werden. Nachteile sind allerdings die trotz der Sortierungen der Elemente vorhandenen Überlappungen der Volumina sowie der Aufwand der Sortierungen selbst. Da sich in einer dynamischen Szene die Positionen der Elemente ständig ändern, müsste hierbei pro Frame eine Sortierung der Elemente erfolgen. Dies ist bei der Größe heutiger Szenen nicht in Echtzeit realisierbar.

4.6 Hidden Surface Removal und Visibility Culling

Das Problem der Sichtbarkeitsbestimmung ist eines der klassischen Forschungsthemen innerhalb der Computer Grafik überhaupt. Angenommen eine Szene S sei über eine Menge von Primitiven P_i als $S = \{P_0, \ldots, P_n\}$ mit $i = 0, \ldots, n$ beschrieben. Weiterhin sei die Position, die Blickrichtung und das Sichtfeld eines Betrachters B der Szene gegeben. Gesucht ist dann in Bezug auf B eine Menge $F_B = \{F_0, \ldots, F_k\}$ von Fragmenten F_j mit $j = 0, \ldots, k$, deren Abbildung auf die Projektionsebene des Betrachters B von keinem anderen Fragment geschnitten oder überlagert wird. Konsequenterweise muss es sich bei den resultierenden Fragmenten nicht um die ursprünglichen Primitive handeln. Die Motivation hinter der Bestimmung von F_B liegt darin begründet, dass in vielen Situationen F_B deutlich weniger Elemente beinhaltet als S. In diesem Zusammenhang ist vor allem der Begriff der Verdeckungstiefe oder auch Tiefenkomplexität einer Szene von Bedeutung, welcher ein Maß repräsentiert, wieviele Primitive ein gegebenes Primitiv P verdeckt. Die Verdeckungstiefe ist allerdings stets in Kombination mit dem Betrachter zu verstehen, da sie etwa bei einer diagonalen Draufsicht auf eine Szene deutlich geringer ausfällt als bei einem Betrachterstandort inmitten der Szene. Diese Abhängigkeit verdeutlicht eine der großen Herausforderungen der Sichtbarkeitsbestimmung, nämlich dass bereits geringe Änderungen der Betrachterattribute

große Auswirkungen auf F_B haben können. Häufige Beispiele für massiv verdeckte beziehungsweise *high densely occluded* Szenen sind Stadtmodelle oder Innenraumarchitekturen. Letztere haben insbesondere den Begriff der Portale geprägt, wobei es sich um Löcher innerhalb der Szene wie etwa geöffnete Türen oder Fenster handelt. Blickt ein Betrachter durch eine Türöffnung in einen Nachbarraum, so ist sein Blickfeld durch den Türrahmen und die Tür selbst begrenzt.

In den Anfängen der Sichtbarkeitsberechnung wurden sogenannte *Hidden Surface Removal* (HSR) Algorithmen entwickelt, welche auch unter der Bezeichnung *Visible Surface Determination* Verfahren bekannt sind. Diese Kategorie von Algorithmen bestimmt nicht nur die Menge der sichtbaren Primitive V_B, sondern berechnet die exakten Fragmente, welche von einem sichtbaren Primitiv verbleiben.

Definition (Exakte Sichtbarkeitsmenge): Die exakte Sichtbarkeitsmenge V_B, auch *Visibility Set* genannt, ist die Menge aller Primitive, welche in Bezug auf einen Betrachter B zumindest teilweise sichtbar sind.

Grundsätzlich lassen sich Hidden Surface Removal Techniken in objektpräzise, bildpräzise und hybride Methoden gliedern. Objektpräzise Ansätze vergleichen die Primitive einer Szene untereinander, um die sichtbaren Bereiche zu bestimmen. Ein Verfahren hierfür wurde von Weiler und Atherton vorgestellt [WA77]. Leider sind objektpräzise Methoden aufgrund der begrenzten Rechengenauigkeit der Computer kaum robust zu implementieren und erfordern zudem eine hohen Rechenaufwand zu Lasten des Hauptprozessors. Bildpräzise Verfahren gehen von der diskreten Auflösung des resultierenden Bildes aus und bestimmen für jeden Pixel das sichtbare Fragment. Beispiele für bildpräzise Ansätze sind die *Ray Casting* Methode von Appel [App68], die *Scan-Line* Methode von Bouknight [Bou70] beziehungsweise Watkins [Wat70] sowie der *z-Buffer* von Catmull [Cat74]. Letzterer ist durch seine einfache Implementation in Hardware zu einem Standard innerhalb der Sichtbarkeitsbestimmung geworden. Hybride Ansätze kombinieren objektpräzise und bildpräzise Methoden. Hierzu sind vor allem Verfahren zu zählen, welche eine Prioritätsliste und somit eine Reihenfolge bestimmen, in der die Primitive einer Szene zu rendern sind. Die hierbei übliche *Back-to-Front* Sortierung basiert auf der Definition, dass ein Vorgängerelement innerhalb der Liste nur von seinen Nachfolgern verdeckt werden kann. Eine positive Eigenschaft derartiger Ansätze ist die Unterstützung transparenter Primitive. Bekannte Ansätze stammen von Schumacker et al. [SBGS69], Newell et al. [NNS72] und Fuchs et al. [FKN80].

Ein großer Nachteil vieler der oben aufgeführten Verfahren ist die redundante Rasterisierung identischer Bildbereiche. Außerdem muss für eine korrekte Visualisierung jedes Primitiv betrachtet werden, weshalb die Laufzeit der Verfahren superlinear zur Anzahl der Elemente in S ist. Im Gegensatz dazu versuchen *Visibility Culling* Verfahren, das Rendern nicht sichtbarer Primitive zu vermeiden und bestimmen hierfür die sichtbare Menge V_B der Primitive. Ein einfaches Beispiel ist das *Backface Culling*, bei dem vom Betrachter abgewandte Primitive bei der Visualisierung nicht berücksichtigt werden. Derartige Primitive können mit Hilfe des Skalarprodukts einfach identifiziert werden, da ihre Normalen entgegengesetzt zum Betrachterpunkt weisen. Ein sublinearer Ansatz hierzu wurde von Kumar et al. [KMGL99] präsentiert. Im Rahmen des *View Frustum Cullings* werden alle Primitive identifiziert, welche außerhalb der Sichtpyramide des Betrachters liegen und somit unsichtbar sind. Übliche

Vorgehensweisen machen dabei von den in den Abschnitten 4.4 und 4.5 beschriebenen hierarchischen Datenstrukturen Gebrauch, um eine schnelle Identifizierung ganzer unsichtbarer Bereiche einer Szene zu ermöglichen [AM00, Cla76, SC96].

Als eine weitere Art des Visibility Cullings werden *Occlusion Culling* Verfahren als ausgabesensitiv oder auch *output sensitive* erachtet, d.h. ihre Laufzeit ist nicht superlinear zur Anzahl der Elemente in S, sondern proportional zur Anzahl der sichtbaren Primitive in V_B. Dabei sind zwei weitere Arten der Sichtbarkeitsmenge zu berücksichtigen, nämlich die konservative Sichtbarkeitsmenge K_B und die approximative Sichtbarkeitsmenge A_B. Erstere beinhaltet die exakte Sichtbarkeitsmenge V_B und klassifiziert zusätzlich nicht sichtbare Primitive als sichtbar, allerdings nicht umgekehrt. Letztere umfasst einige Primitive aus V_B und zusätzlich einige weitere unsichtbare Primitive, welche als sichtbar deklariert wurden.

Definition (Konservative Sichtbarkeitsmenge): Die konservative Sichtbarkeitsmenge K_B beinhaltet die Menge aller in Bezug auf einen Betrachter B sichtbaren Primitive V_B plus einiger weiterer unsichtbarer Primitive, welche zuvor als sichtbar identifiziert wurden.

Definition (Approximative Sichtbarkeitsmenge): Die approximative Sichtbarkeitsmenge A_B beinhaltet eine Teilmenge aller in Bezug auf einen Betrachter B sichtbaren Primitive V_B plus einiger weiterer unsichtbarer Primitive, welche zuvor als sichtbar identifiziert wurden.

Anstelle der exakten Sichtbarkeitsmenge V_B bestimmen viele Occlusion Culling Verfahren lediglich die konservative Sichtbarkeitsmenge K_B oder die approximative Sichtbarkeitsmenge A_B und überlassen die exakte Auflösung des Sichtbarkeitsproblem dem z-Buffer. Die Anzahl der Verfahren, welche auf der Bestimmung von A_B basieren ist hierbei deutlich geringer [BMH98, BMH99, KS99, KS00, ZMHHI97, Zha98]. In der Regel werden im Bildbereich sehr kleine Primitive beziehungsweise Regionen vernachlässigt. Aufgrund der heutigen Realisierung von z-Buffer, Backface Culing und View Frustum Culling in Hardware gibt es ein wichtiges Kriterium für die Güte eines Occlusion Culling Verfahrens: Der Aufwand der Bestimmung von V_B, K_B oder A_B plus der Visualisierungsaufwand der entsprechenden Menge muss (deutlich) geringer sein, als die Menge S schlicht und einfach an die Grafik-Hardware zu übergeben. Dies stellt bei den aktuellen Grafikchips durchaus ein schwieriges Unterfangen dar.

Um das Ziel der Ausgabesensitivität zu erreichen, werten Occlusion Culling Verfahren drei verschiedene Arten von Kohärenzen aus, nämlich räumliche Kohärenzen, bildbasierte und temporäre Kohärenzen. Räumliche Kohärenzen basieren auf der räumlichen Strukturierung der Szene durch einen Raumunterteilungsbaum oder einer Bounding Volume Hierarchy, welche benachbarte Primitive in entsprechende Zellen zusammenfassen. Wird beispielsweise eine Zelle eines Raumunterteilungsbaumes als unsichtbar erkannt, so impliziert dies automatisch die Unsichtbarkeit aller in der Zelle liegenden Primitive und Unterzellen. Entsprechende Tests konvertieren üblicherweise die Begrenzungen der Zelle in eine Menge von Primitiven und überprüfen diese auf Sichtbarkeit. In Kombination mit Sichtbarkeitstest wird daher im folgenden eine Zelle stets als eine solche Menge von Primitiven verstanden. Bildbasierte Kohärenzen nutzen die Nachbarschaft der Primitive im Bildbereich aus. Ein Beispiel hierfür ist der hierarchische z-Buffer [GKM93, GK94], bei dem analog zum Schema eines regelmäßigen Quadtrees jeweils immer die z-Werte vier benachbarter Pixel beziehungsweise Zellen zum größten z-Wert zusammengefasst werden, um hieraus eine hierarchische z-Pyramide

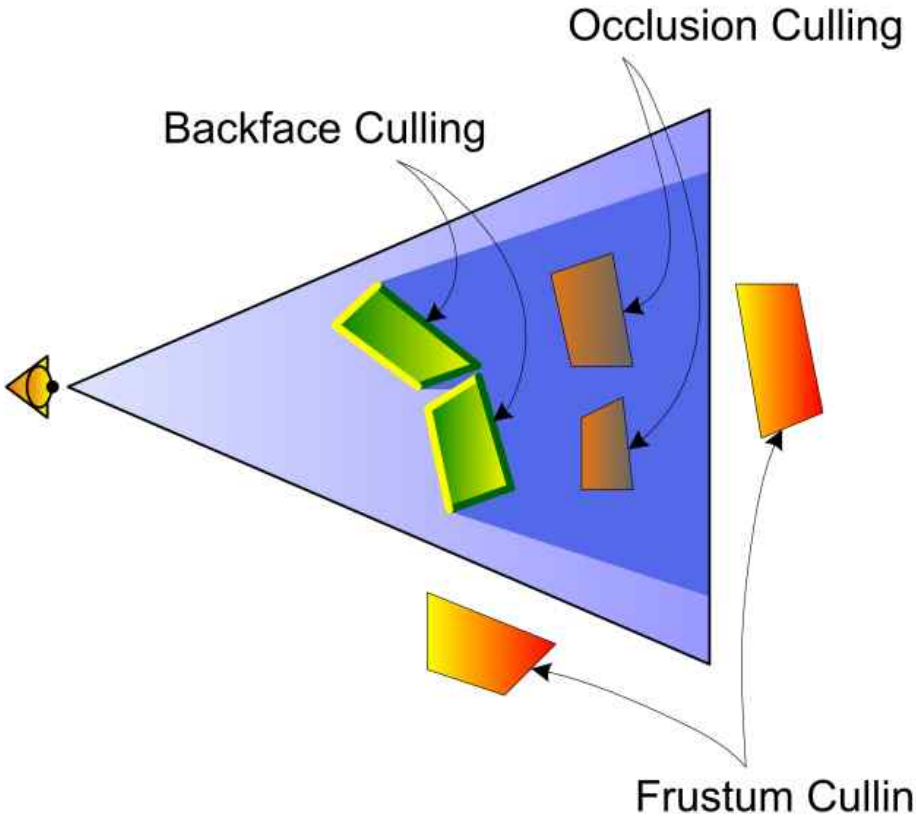

Abbildung 4.15: *Im einfachen Fall des Backface Cullings werden die rückseitigen, d.h. die vom Betrachter abgewandten Flächen, geschlossener Modelle entfernt. Während View Frustum Culling Verfahren sich auf die Identifikation der Elemente innerhalb der Sichtpyramide beschränken, berücksichtigen Occlusion Culling Konzepte zusätzlich die Verdeckungen der Elemente untereinander.*

zu erstellen. Temporäre Kohärenzen gehen von einer stetigen Bewegung des Betrachters und eventueller dynamischer Objekte innerhalb der Szene aus, d.h. es erfolgen keine abrupten Richtungs- oder Geschwindigkeitsänderungen. Unter diesen Umständen sollten die Änderungen zwischen zwei Frames äußerst gering ausfallen, weshalb die bestimmte Sichtbarkeitsmenge mit hoher Wahrscheinlichkeit entweder weiterhin gültig bleibt oder nur geringen Abweichungen unterliegt.

Eine Taxomierung der einzelnen Verfahren ist auf vielfältige Art und Weise möglich, beispielsweise über die Art der bestimmten Sichtbarkeitsmenge, über die ausgewerteten Kohärenzen, über die Unterstützung dynamischer Szenen oder über die Beanspruchung aktuell vorhandener Hardware. Ein weiteres Kriterium ist die Verwendung von *Occludern*. Hierbei handelt es sich um Primitive, welche in Bezug auf einen Betrachter den Großteil einer Szene verdecken. Von Occludern verdeckte Primitive werden als *Occludees* bezeichnet. Das Prinzip der Occluder beruht auf der Beobachtung, dass sich hierfür in der Regel nur einige wenige Primitive verantwortlich zeichnen. Als Konsequenz handelt es sich häufig um große Primitive in der Nähe des Betrachters. Verfahren, die von Occludern Gebrauch machen, bestimmen zunächst im Rahmen einer Occluder-Selektion die Menge der Occluder. Um nicht die gesamte Szene auf Occluder überprüfen zu müssen, werden potentielle Occluder einer Szene oftmals in einem Vorverarbeitungsschritt identifiziert und in einer räumlich sortier-

ten Occluder-Datenbank abgelegt. Coorg et al. [CT97] präsentieren eine einfache Formel zur dynamischen Bestimmung der Occluder:

$$\delta = \frac{-A(\vec{N} \cdot \vec{V})}{\left\|\vec{D}\right\|^2} \tag{4.9}$$

In dieser Formel beschreibt A den Flächeninhalt eines Primitivs, $\vec{N}$ dessen Normale, $\vec{V}$ die Blickrichtung des Betrachters und $\vec{D}$ den Vektor von der Betrachterposition zum Zentrum des Primitivs. Überschreitet δ einen bestimmten Grenzwert, dann ist das Primitiv als Occluder geeignet. Nach der Auswahl der Occluder werden die restlichen Primitive der Szene gegen die Occluder getestet. Dabei sind viele Parallelen zu Schattenwurf-Verfahren [WPF90] zu erkennen, da etwa Primitive, die vollständig im Schattenwurf eines Occluders liegen (mit dem Betrachter als Lichtquelle) nicht sichtbar sind. Weil eine derartige Vorgehensweise mit steigender Zahl der Occluder einen immensen Aufwand beansprucht, wurden *Occluder Fusion* Techniken zur Zusammenfassung mehrerer Occluder zu einer einzelnen Occluder-Repräsentation entwickelt [GKM93, ZMHHI97, BHS98, KS00].

Ein weitere Gruppe von Verfahren basiert auf dem Prinzip der *Potentially Visible Sets* (PVS). Hierbei erfolgt zuerst eine Unterteilung der Szene in mehrere Zellen und daraufhin für jede Zelle eine Sichtbarkeitsanalyse, deren Ergebnisse in der jeweiligen Zelle gespeichert werden. Unter der Annahme, dass ein Betrachter innerhalb der Zelle bestimmte Positionen und Blickrichtungen einnehmen kann, ist das Ziel der Analyse üblicherweise die Bestimmung einer konservativen Sichtbarkeitsmenge. Da eine entsprechende Analyse einen hohen Rechenaufwand erfordert, wird selbige einschließlich der Partitionierung in einem Vorverarbeitungsschritt durchgeführt. Verfahren dieser Art verfügen in der Regel über geringe Laufzeiten, da lediglich in Abhängigkeit von der Position des Betrachters die Informationen der betretenen Zelle abzurufen sind. Allerdings kommen sie fast ausschließlich in statischen Szenarien zum Einsatz, weil ansonsten wieder eine teure Vorberechnung vonnöten ist. Teller et al. [TS91] beschreiben eine 2D Implementierung für Innenräume, welche über die Einschränkung von Portalen die Menge aller von einer bestimmten Zelle aus sichtbaren Zellen bestimmt. Teller et al. berücksichtigen weder die genaue Position eines Betrachters innerhalb einer Zelle noch dessen Blickrichtung, weshalb die resultierende konservative Sichtbarkeitsmenge immer noch sehr groß ausfällt. Weitere Ansätze stammen unter anderem von Plantinga [Pla93], Saona-Vazquez et al. [SVNB99] und Cohen-Or et al. [COFHZ98a]. Koltun et al. [KCCO00] beschäftigen sich in ihrem Verfahren mit dem nicht zu unterschätzenden Problem des hohen Speicherbedarfs von PVS-basierten Ansätzen. In der Regel beinhalten nämlich die analysierten Sichtbarkeitsmengen zusammen deutlich mehr Elemente als die Szene S selbst.

Leider existieren nur sehr wenige Verfahren, die dynamische Szenen unterstützen. Eine Begründung hierfür ist, dass sehr viele Ansätze Gebrauch von mehr oder weniger teuren Vorberechnungen machen, um darauf eine interaktive Visualisierung in Echtzeit anbieten zu können. Eine genaue Definition, was nun wirklich als Vorberechnung zu zählen ist, fällt dabei schwer. Weil viele Algorithmen von der räumlichen Sortierung einer Szene abhängig sind, kann auch deren Bestimmung als eine Art der Vorberechnung erachtet werden. Im Rahmen dieser Arbeit wird allerdings davon ausgegangen, dass zumindest im Falle der Raumunter-

teilungsbäume eine Lösung für die konsistente Pflege der räumlichen Kohärenzen in Echtzeit existiert (siehe Abschnitt 4.4). Insofern sind hier nur aufwändigere Prozeduren wie etwa die Sichtbarkeitsanalyse innerhalb PVS-basierter Verfahren im Bereich von mehreren Minuten bis Stunden als Vorberechnung zu werten. Für das Problem der ausgabesensitiven Visualisierung beliebiger dynamischer 3D Szenen erscheint es beinahe unerlässlich, die gegebene Grafik-Hardware auszunutzen, um auch ohne Vorberechnungen interaktive Frameraten zu erreichen. In den letzten Jahren hat sich das Spektrum der von der Hardware unterstützten Features extrem erweitert. So bieten alle neueren Grafik-Chips der NVIDIA- oder Ati-Reihe das *HP-Flag* [HPF] und die *NVIDIA-Occlusion-Query* an. Das HP-Flag gibt für jedes gerenderte Primitiv ein Signal zurück, ob bei dem Rendervorgang der z-Buffer verändert wurde oder nicht. Ist das Flag nicht gesetzt, so ist das Primitiv unsichtbar. Dieses Prinzip kann auch auf die virtuellen Zellen der Raumunterteilungsbäume angewendet werden, so dass hier schnelle Sichtbarkeitstests möglich sind. Die NVIDIA-Occlusioin-Query ist eine Erweiterung des HP-Flags und gibt nicht nur ein einzelnes Bit, sondern die Anzahl der wirklich gesetzten Pixel eines gerenderten Primitives zurück. Hierdurch ist zum einen die elegante Berechnung der approximativen Sichtbarkeitsmenge A_B, zum anderen aber auch die Bestimmung der beim Rendern zu verwendenden Detailstufe eines Modells möglich. Ein für die genannten Hardware-Erweiterungen geeignetes Verfahren stammt von Bartz et al. [BMH99]. Zhang et al. [ZMHHI97] beschreiben einen weiteren Hardware-basierten Ansatz, welcher Parallelen zu [GKM93, GK94] beinhaltet und auf der *Mipmapping* Architektur von OpenGL beruht.

4.7 Szenegraph

Nach den Abschnitten 4.4 und 4.5 können Raumunterteilungsbäume und Bounding Volume Hierarchien für die räumliche Sortierung der Elemente einer Szene eingesetzt werden. Dies alleine ist aber noch nicht ausreichend, um eine 3D Szene vollständig zu beschreiben. Beispielsweise fehlt noch die Repräsentation der Elemente selbst, d.h. ihr äußeres Erscheinungsbild sowie ihr Verhalten. Wie in Abschnitt 4.2 erläutert, stellt hierfür der Ansatz der Animationselemente eine Option dar. Es gibt aber noch weitere Informationen wie etwa Lichtquellen oder Hintergrundfarben, welche ebenfalls Teil einer Szene sind. Aus diesem Grund werden für die Repräsentation einer Szene sogenannte *Szenegraphen* eingesetzt, die sämtliche Informationen einer Szene beinhalten.

Wie der Name schon sagt, entspricht ein Szenegraph einem Graphen aus Knoten und Kanten. Jeder Knoten repräsentiert einen bestimmten Typ und damit eine bestimmte Informationsmenge, die er zur Darstellung der Szene beisteuert. Beispielsweise gibt es Knoten zur Beschreibung der Geometrie, der Farben oder der Texturen eines Elements. Andere Knoten wiederum sind in der Lage, bestimmte Konstellationen einer Szene zu erkennen und in Form von Ereignissen an interessierte Knoten weiterzuleiten. Die Kanten des Szenegraphen beschreiben dabei die Zusammenhänge und Beziehungen zwischen den einzelnen Knoten. Hierüber können etwa die Farben einer Geometrie oder eine Aktion einer bestimmten Situation zugeordnet werden. In vielen Szenegraphen geht mit diesem Konzept ein wenig der objektorientierte Gedanke verloren. Im Gegensatz zum Ansatz der Animationselemente ist es nämlich oftmals schwierig, Verhalten und äußeres Erscheinungsbild aus der Darstellung

Szenegraph	**Referenz**
VRML (X3D)	http://www.web3D.org
Open Inventor	http://www.sgi.com/software/inventor
IRIS Performer	http://www.sgi.com/software/performer
Coin3D	http://www.coin3d.org
OpenSG	http://www.opensg.org
Direct3D	http://www.microsoft.com/windows/directx
Java3D	http://java.sun.com/products/java-media/3D
OpenRM Scene Graph	http://www.openrm.org
OpenSceneGraph	http://openscenegraph.sourceforge.net/index.html
Jupiter	[BSS+01]

Tabelle 4.3: *Eine Auflistung einiger aktueller Szenegraphen.*

des Szenegraphen mit einem konkreten Element in Zusammenhang zu bringen.

Wie in Tabelle 4.3 zu erkennen, gibt es eine Vielzahl verschiedener Szenegraphen. In der Regel zeichnet sich ein Szenegraph von den jeweils anderen durch ein besonderes Merkmal aus. So legen etwa IRIS Performer und OpenSG einen Schwerpunkt auf die Fähigkeit zur parallelen Abarbeitung. An dieser Stelle würde eine Vorstellung der einzelnen Szenengraphen zu weit führen. Vielmehr ist hier eine häufige Gemeinsamkeit der Szenegraphen von Bedeutung, nämlich die Art und Weise, wie derartige Graphen interpretiert werden.

Während der Abarbeitung eines Szenegraphen werden seine Knoten traversiert und mit jedem Betreten eines Knotens dessen spezifische Informationen als aktueller Zustand gesetzt. Dieser Zustand bleibt solange erhalten, bis er durch den Besuch eines anderen Knotens überschrieben wird. Eine Voraussetzung für dieses Prinzip ist ein zugrunde liegendes Zustandsmodell, wie es etwa OpenGL [SGIc] anbietet. Setzt beispielsweise ein Knoten die aktuelle Zeichenfarbe F, so erfolgt eine Neudefinition dieses Wertes erst mit dem Erreichen eines weiteren Farbknotens. Allen dazwischen spezifizierten Geometrien wird die Farbe F zugewiesen.

4.8 Netzwerktopologien

Analog zu Polygonnetzen haben auch verteilte Anwendungen wie etwa virtuelle Umgebungen einen Bezug zur Graphentheorie. Während die Knoten des Graphen den Teilnehmern der Applikation entsprechen, repräsentieren die Kommunikationskanäle zwischen den Teilnehmern die Kanten und somit die Topologie des Graphen. Hierbei ist zwischen zwei grundlegenden Strukturen zu unterscheiden. Während Peer-to-Peer Konzepte die Gleichberechtigung aller Teilnehmer propagieren und daher jeder Knoten mit allen anderen Knoten des Graphen kommunizieren darf, erlauben Client-Server Systeme lediglich die Kommunikation zwischen Client und Server. Die gewählte Topologie hat entscheidenden Einfluss auf den Kommunikationsaufwand, der beispielsweise mit der Synchronisation der dynamischen Szenen anfällt.

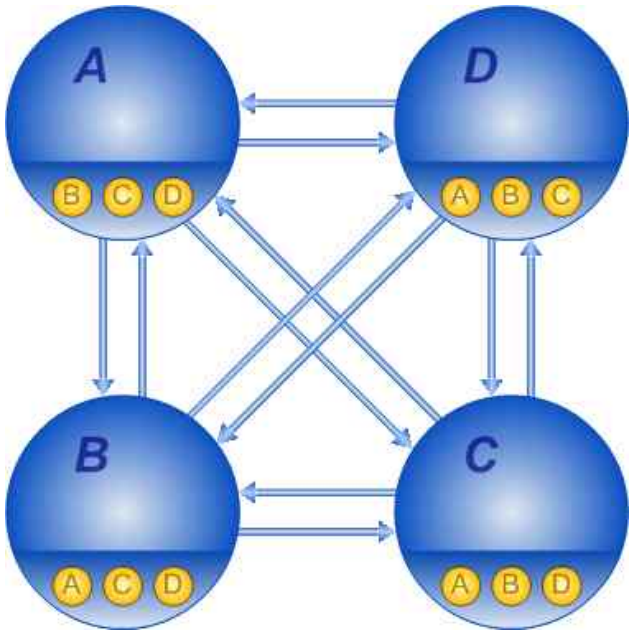

Abbildung 4.16: *Im Fall einer Peer-to-Peer Topologie fallen pro Zeitschritt schlimmstenfalls n^2 Nachrichten an, sofern das zugrunde liegende Netzwerk ausschließlich Unicast Verbindungen unterstützt. Die orange gefärbten Bezeichner symbolisieren die auf einem Knoten potentiell sichtbaren Teilnehmer.*

Im optimalen Fall sollte eine durch einen Teilnehmer vorgenommene Manipulation der Szene allen Benutzern einschließlich dem Verursacher gleichzeitig mitgeteilt werden. Die folgenden Klassifizierungen der Topologien hinsichtlich der Komplexität des Kommunikationsaufwands beruhen auf einem Szenario, in dem der Graph einer verteilten 3D Simulation aus n Knoten beziehungsweise Teilnehmern besteht. Jeder Teilnehmer ist durch einen Avatar repräsentiert. Die Avatare entsprechen den einzigen dynamischen Elementen der durch die Simulation beschriebenen 3D Szene und können während der Navigation der Teilnehmer in den Sichtbereich anderer Benutzer gelangen. Aus diesem Grund sind ihre Animationen auf den einzelnen Knoten zu synchronisieren. Die Simulation ist durch Zeitschritte diskretisiert, wobei mit jedem Zeitschritt eine Aktualisierung der Avatare anfällt.

Die Systeme *Reality Built for Two* [BGH+90], *VEOS* [BC93], *MR Toolkit* [SG93] sind Beispiele für Peer-to-Peer Konzepte, die auf einem in Abbildung 4.16 dargestellten *Unicast* Netzwerk basieren. Hier müssen im Falle der Transformation eines Avatars $n-1$ Nachrichten an die anderen Knoten versendet werden. Die Bewegung aller n Teilnehmer während eines Zeitschritts der Simulation resultiert in n^2 Nachrichten, was der Komplexität $O(n^2)$ entspricht. Aus diesem Grund sind die genannten Systeme hinsichtlich der Teilnehmerzahl stark eingeschränkt.

Demgegenüber erfordern Peer-to-Peer Ansätze wie etwa *SIMNET* [CDG+93] oder *VERN* [BHML92] lediglich eine Komplexität von $O(n)$, da sie auf einem *Broadcast* Netzwerk basieren. Derartige Netzwerke versenden eine einzelne Nachricht automatisch an alle übrigen Teilnehmer, weshalb die Knoten pro Zeitschritt bei der Transformation sämtlicher Avatare insgesamt n Nachrichten verschicken müssen. Der Empfang einer Nachrichten auf einem Knoten resultiert in einer Aktualisierung des mit der Nachricht assoziierten Avatars. Da-

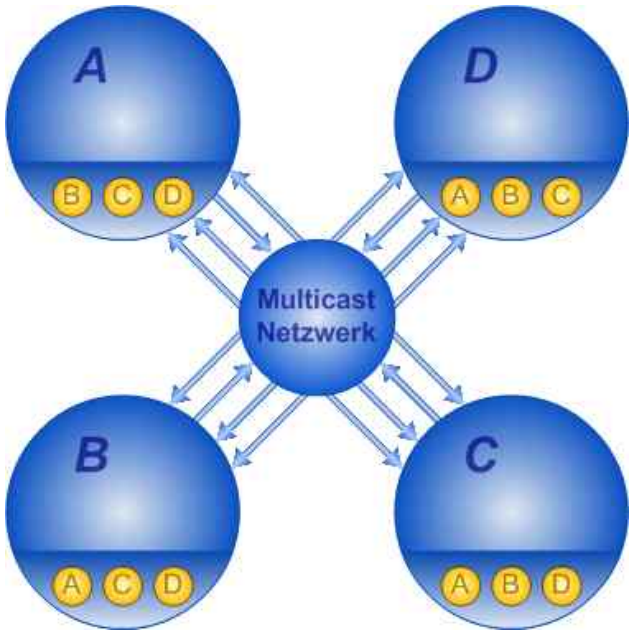

Abbildung 4.17: *Im Gegensatz zu Broadcast Netzwerken verteilen Multicast Netzwerke die Nachrichten nicht an alle Benutzer, sondern lediglich an die Knoten innerhalb der Gruppe des Absenders.*

her ist für jeden Knoten die Bearbeitung von $n-1$ Avataren erforderlich, was immer noch einem immensen Aufwand gleichkommt. Um diesem Problem zu begegnen, verwenden die Peer-to-Peer Systeme *DIVE* [CH93a, CH93b, Hag96, FS98] und *NPSNET* [ZPF+93] ein in Abbildung 4.17 dargestelltes *Multicast* Netzwerk. Deren Idee beruht darauf, die Benutzer nach bestimmten Kriterien in Gruppen aufzuteilen. Im oben genannten Szenario könnten beispielsweise räumlich benachbarte Teilnehmer zusammengefasst werden. Der Nachrichtenaustausch ist dann nur noch zwischen den Partnern einer Gruppe erforderlich.

Peer-to-Peer Ansätze sind insgesamt betrachtet aufwändiger zu synchronisieren als Client-Server Systeme, da sie im Gegensatz zu letzteren keine zentrale Beschreibung einer Szene aufweisen. Im Falle eines Client-Server Systems wie etwa *RING* [Fun95], *WAVES* [Kaz93] oder *BrickNet* [GSPN95] übermitteln alle Teilnehmer eine Nachricht an den Server, sofern zuvor eine Transformation des jeweiligen Avatars erfolgt ist. Insgesamt versenden die n Teilnehmer für einen Zeitschritt also immer noch n Nachrichten. Jedoch kann der Server seine zentrale Szenenbeschreibung ausnutzen, um die weitergeleiteten Nachrichten zu filtern. Konsequenterweise erhalten nur diejenigen Teilnehmer eine Nachricht, die auch ein Interesse an ihr haben. Das Problem der Client-Server Ansätze beruht auf der hohen Belastung des Servers durch die Evaluation und das Weiterleiten der Nachrichten. Abbildung 4.18 zeigt daher eine alternative Vorgehensweise in Form einer Hierarchie von Servern [Fun96b]. Diese Server sind über eine schnelle Verbindung mit hoher Bandbreite vernetzt, wie sie beispielsweise in einem *Local Area Network* (LAN) oder Cluster gegeben ist. Während einer der Server nach wie vor die 3D Szene verwaltet, übernehmen die restlichen Server das Weiterleiten der Nachrichten. Hierdurch entstehen bei der Synchronisation der dynamischen Szenen eventuell höhere Latenzzeiten als bei einem einzelnen Server, da der empfangende Server zunächst seine Szene gemäß der Nachricht aktualisiert und erst im Anschluss an einen Nachrichtenserver übergibt.

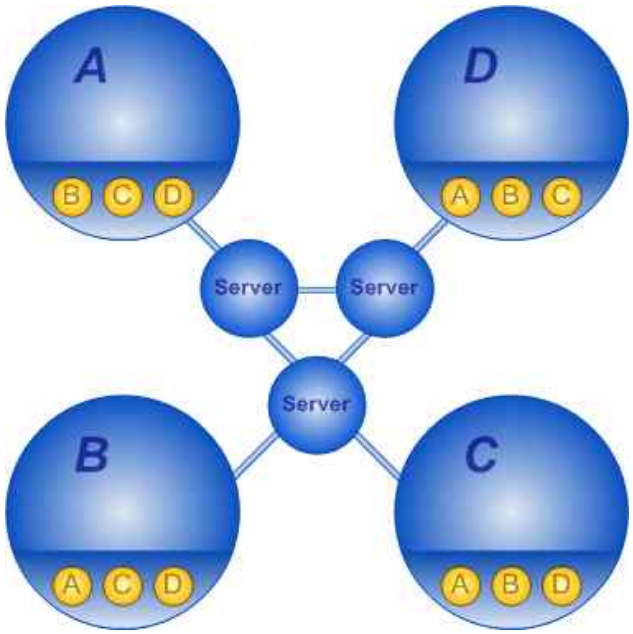

Abbildung 4.18: *In Client-Server Systemen erfolgt die Kommunikation ausschließlich über Client-Server Verbindungen. Hierarchische Server Konzepte dienen der Entlastung eines einzelnen Servers beispielsweise durch spezialisierte Nachrichtenserver.*

4.9 Zusammenfassung

In diesem Kapitel wurden wichtige Techniken beschrieben, die für eine Umsetzung der in Kapitel 2 aufgestellten Anforderungen erforderlich sind. Um nach Anforderung 2.1 eine Szene repräsentieren zu können, werden Techniken zur Beschreibung der grundlegenden Bausteine einer Szene benötigt. Der Begriff der Elemente steht stellvertretend für derartige Bausteine. Ein Element kann sowohl ein einzelnes Primitiv wie etwa eine Linie, ein Polygon, oder einen Zylinder darstellen als auch in Form eines komplexen Elements aus mehreren Primitiven bestehen. Für die Repräsentation eines Elements gibt es eine Vielzahl verschiedener Techniken, beispielsweise unstrukturierte Punkt- oder Polygonwolken sowie flächen- beziehungsweise volumenbasierte Beschreibungen. Innerhalb dieser Arbeit sind vor allem flächenbasierte Repräsentationen von Bedeutung, welche sich ebenfalls in mehrere Teildisziplinen gliedern. Die vielleicht am häufigsten frequentierte Vorgehensweise sind Polygonnetze, auch Meshes genannt, welche eine intuitive und einfache Form der Modellierung ermöglichen. Die spezialisierte Form der Dreiecksnetze hat den Vorteil, sich aus den stets planaren und konvexen Dreiecken zusammenzusetzen und zudem die Unterstützung der Grafik-Hardware zu finden. Leider können Polygonnetze einen gekrümmten Körper nur annähern, weshalb für gute Approximationen ein enormer Datenaufwand anfällt. Da Polygonnetze letztlich einen Graphen bestehend aus Scheitelpunkten und deren Konnektivität darstellen, können die aus der Graphentheorie bekannten Definitionen wie etwa die Orientierbarkeit, das Geschlecht oder die Zusammenhangskomponenten übernommen werden. Die Euler-Poincaré-Formel liefert eine nützliche Abschätzung für das Verhältnis zwischen der Anzahl der Scheitelpunkte und der Anzahl der Polygone in einem Polygonnetz. Je nach Anwendungsbedarf gibt es mit punkt-, kanten- sowie polygonbasierten Ansätzen verschiedene effiziente Möglichkeiten,

ein Polygonnetz zu beschreiben. Ausschlaggebend für die Wahl eines Ansatzes ist nicht nur dessen Speicherverbrauch, sondern auch die schnelle Verfügbarkeit aller Informationen des Polygonnetzes.

Eine andere flächenbasierte Art der Beschreibung von Elementen ist unter dem Begriff der Freiformflächen beziehungsweise -kurven bekannt. Sie bieten insbesondere für gekrümmte Bereiche eine kompakte und effiziente Darstellung. Unglücklicherweise müssen sie vor einer Visualisierung zumeist in eine von der Grafik-Hardware unterstützte, polygonbasierte Repräsentation konvertiert werden, womit ein erheblicher Aufwand entsteht. Grundsätzlich ist zwischen der expliziten, der impliziten und der parametrischen Darstellung zu unterscheiden. In der expliziten Form $y = f(x)$ darf es für jeden Wert x auch nur jeweils einen Wert y geben. Zudem ist die explizite Beschreibung nicht invariant gegenüber Rotationen und kann keine Kurven mit wirklich vertikalen Tangenten repräsentieren. Eine implizite Geometrie wird definiert durch eine Basisgeometrie B, eine Distanzfunktion $d_b(P)$ und einen Schwellwert beziehungsweise Isowert $s > 0$. Im dreidimensionalen Raum R^3 ist die implizite Geometrie durch die Menge M_s der Punkte $P \in R^3$ gegeben, für die $d_b(P) = s$ ist. Bei dieser Darstellung besteht die Möglichkeit mehrerer Lösungen, weshalb für ein eindeutiges Ergebnis zusätzliche Randbedingungen erstellt werden müssen. Ein weiteres Problem der impliziten Beschreibung ist die oftmals nicht triviale Kalkulation der Tangenten. Demgegenüber bieten parametrische Repräsentationen den Vorteil der Invarianz hinsichtlich Rotationen sowie eine Lösung für das Problem der unendlichen Steigungen in Form der Tangentenvektoren. Parametrische Repräsentationen sind in der Regel durch ganz rationale oder gebrochen rationale Polynome n-ten Grades bestimmt, wobei in der Computer Grafik überwiegend kubische Beschreibungen mit $k = 3$ genutzt werden. Diese stellen die niedrigste Ordnung da, um eine Kurve über zwei Endpunkte und deren Steigung zu spezifizieren. Parametrische Repräsentationen lassen sich über exakte, interpolatorische und approximative Darstellungen definieren. Bei der exakten Darstellung ist jeder Punkt durch eine Formel definiert, wobei letztere jedoch zumeist entweder nicht bekannt oder zu komplex ist. Die interpolatorische Darstellung beschreibt ebenso wie die approximative Form eine Fläche oder Kurve durch Stützstellenbedingungen. Sie weist insbesondere bei hohen Polynomgraden häufig das Problem der Oszillation auf. Im Gegensatz zur interpolatorischen Repräsentation determiniert in der approximativen Form die Fläche oder Kurve nicht an den Stützstellen, wodurch eine intuitive Modellierung erschwert wird. Für beide Vorgehensweisen existiert eine Vielzahl verschiedener Ansätze, von denen vor allem die approximative Beschreibungsform der Splines hervorzuheben ist. Splines bieten an Segmentübergängen eine parametrische C^2 Stetigkeit und verlaufen dort daher glatter als andere Repräsentationen. Als wichtige Unterart der Splines bieten die Nonuniform Rational B-Splines (NURBS) nicht nur die Invarianz bezüglich Translation, Rotation und Skalierung, sondern auch bezüglich perspektivischer Transformationen.

Während Polygonnetze und Freiformflächen ausschließlich das visuelle Erscheinungsbild der Elemente einer Szene beschreiben, erlaubt das Konzept der Animationselemente eine Verknüpfung zwischen der visuellen und der verhaltensspezifischen Beschreibung eines Elements. Demnach ist ein Animationselement als ein unabhängiger, gekapselter Baustein definiert, der zur Erzeugung dynamischer Szenarien verwendet wird. Er beinhaltet sowohl die Beschreibung der visuellen Erscheinungsform des korrespondierenden Elements als auch Wissen über dessen spezifisches Verhalten.

Hinsichtlich der nach Anforderung 2.3 verlangten Adaptivität der Daten stellen Level-of-Detail Konzepte eine Lösungsmöglichkeit dar. Hierbei wird ein Element in verschiedenen Detailstufen gespeichert und anhand bestimmter Kriterien eine geeignete Stufe zur Laufzeit ausgewählt. Die ursprüngliche Intention der Level-of-Detail liegt in der Reduktion des Visualisierungsaufwandes. Elemente, die sich etwa in großer Entfernung zum Betrachter befinden, können in einem sehr einfachem Detailgrad dargestellt werden. Neben der Distanz zum Betrachter sind andere Kriterien die Originalgröße des Elements, die Lage des Elements im Sichtbereich sowie die Dynamik von Element und Betrachter. Für die automatische Generierung der Detailstufen eines Elements gibt es eine Vielzahl verschiedener Ansätze, die sich in Verfahren zur Entfernung von Geometrie (Decimation, Simplifizierung), in (Re-) Sampling Ansätze und in Verfeinerungsalgorithmen (Refinement, Subdivision) klassifizieren lassen.

Konzepte der ersten Kategorie beruhen auf der Vereinfachung des ursprünglichen Polygonnetzes $\hat{M} = M^n$ in das nicht weiter zu simplifizierende Basismesh M^0. Dazu werden n Transformationen benötigt, die jeweils den Übergang eines Meshes M^i in das vereinfachte Mesh M^{i-1} beschreiben. Jede Transformation wird durch einen Operator definiert, der auf das Polygonnetz angewendet wird. Topologische Operatoren können sowohl Auswirkungen auf die lokale als auch auf die globale Topologie eines Meshes haben, wobei erstere die lokale Umgebung der Transformation betrifft und letztere etwa eine Veränderung des Geschlechts bedeutet. Geometrische Operatoren nehmen keine Änderungen an der Topologie vor, sondern verschieben lediglich die Position eines Scheitelpunktes. Die Qualität eines transformierten Meshes wird über ein Qualitätsmaß definiert, das die Güte der Approximation an das Originalmesh $\hat{M}$ beschreibt. Wichtige Begriffe sind in diesem Zusammenhang die lokale und die globale Fehlermetrik. Mit Hilfe der lokalen Fehlermetrik wird der Fehler bestimmt, der durch die Transformation eines Meshes M^i in das vereinfachte Mesh M^{i-1} mit $0 < i \leq n$ entsteht. Dagegen beschreibt die globale Fehlermetrik die Differenz eines Meshes M^i zum Originalmesh $\hat{M}$ und wird aufgrund des hohen Rechenaufwands häufig nur angenähert. Der erstmals progressive Ansatz von Hoppe [Hop96] erzeugt aus einem Mesh $\hat{M}$ das vereinfachte Mesh M^0 und zusätzlich eine Sequenz von Informationen, über die aus M^0 inkrementell wieder $\hat{M}$ rekonstruiert werden kann. Das Verfahren ist daher für die schrittweise Übertragung und Verfeinerung eines Elements auf einem Endgerät von Bedeutung.

Als Verfahren der zweiten Kategorie benötigen Resampling Ansätze weder eine gültige Topologie noch erhalten sie selbige, wodurch sie eine hohe Robustheit gegenüber Sonderfällen aufweisen. In einem ersten Schritt wird mit jedem Scheitelpunkt eines Modells ein Wert assoziiert, welcher die Bedeutung des Punktes hinsichtlich der Charakteristik des Modells beschreibt. Während des zweiten Schrittes erfolgt nach dem Konzept von Rossignac et al. [RB93] die Kalkulation der Bounding Box des Modells sowie deren Unterteilung in regelmäßige Zellen. Anschließend werden in jeder Zelle die dort vorhandenen Scheitelpunkte auf einen einzelnen Punkt kollabiert, welcher anhand des im ersten Schritt ermittelten Wertes bestimmt wird. Wohingegen eine grobe Auflösung der Zellen eine sehr starke Reduktion verursacht, impliziert eine feine Unterteilung lediglich geringe Manipulationen. Resampling Verfahren sind in der Regel einfach zu implementieren, erlauben aber keine Abschätzung der nach der Vereinfachung verbleibenden Dreiecke und sind zudem nicht invariant gegenüber Transformationen. Low et al. [LT97] beschreiben eine verbesserte Variante, die zumindest eine Fehlerabschätzung erlaubt. Anstatt einer regelmäßigen Zellunterteilung berechnen sie

ausgehend von einer Anfangszelle für jeden Reduktionsschritt die folgenden Zellen in Abhängigkeit von ihren Vorgängern.

Unterteilungsflächen (Subdivision Surfaces) entsprechend der dritten Kategorie und erzeugen ausgehend von einem groben Basismesh durch rekursive Verfeinerung (Subdivision) Meshes mit einer höheren Auflösung. Ziel der Verfeinerung ist die Annäherung an das Originalmesh mit Hilfe eines Regelsatzes. Das Ende der Rekursion ist erreicht, sobald das resultierende Mesh entweder eine Mindestzahl an Polygonen überschreitet oder sich vom Originalmesh nur noch durch einen vorgegebenen Schwellwert unterscheidet. Die gundlegende Idee der Unterteilungsflächen wurde ursprünglich von Chaikin vorgestellt [Cha74]. Seitdem hat es viele Arbeiten in diesem Bereich gegeben, die sich in approximierende und interpolierende Verfahren unterteilen lassen.

Beziehen sich die oben aufgeführten Techniken sämtlich auf die Repräsentation eines einzelnen, möglicherweise komplexen Elements, so entsteht nun die Frage nach deren Organisation innerhalb großer 3D Szenen. Anwendungen wie Occlusion Culling Verfahren, Lichtstrahlverfolgungen oder Kollisionserkennungen erfordern eine geeignete räumliche Sortierung der Elemente. Durch die dabei beschriebenen räumlichen Kohärenzen besteht die Möglichkeit verallgemeinernder Aussagen. Ist beispielsweise die konvexe Hülle einer Menge von Elementen für einen Betrachter unsichtbar, so sind auch alle betreffenden Elemente unsichtbar. Einen Lösungsansatz stellen die Raumunterteilungsbäume beziehungsweise Space Partitioning Trees dar, welche erstmals 1969 von Schumacker et al. [SBGS69] in Form der binären Raumunterteilungsbäume zur Sichtbarkeitsbestimmung eingesetzt wurde. Ausgehend von einer die gesamte Szene umfassenden Zelle werden alle Elemente der Szene dieser Zelle zugeordnet. Überschreitet die Zahl der Elemente innerhalb der Zelle eine Kapazität δ, so wird die Zelle durch Wahl einer Hyperebene in zwei Tochterzellen unterteilt und alle Elemente derjenigen Tochterzelle zugeordnet, die sie vollständig beinhaltet. Dadurch entsteht eine hierarchische Struktur, deren Knoten jeweils eine Zelle repräsentieren. Ein ernstes Problem der Raumunterteilungsbäume ist die Bestimmung der Hyperebene, da die resultierende Hierarchie möglichst einem balancierten Baum entsprechen sollte. Eine einfache Lösung beschreiben die k-D-Trees [Ben75, Ben79], welche stets eine achsenparallele, mittige Unterteilung verwenden. Hierdurch tritt verstärkt das Problem der Schnittelemente auf, welche die Hyperebene schneiden und sich somit nicht eindeutig einer Tochterzelle zuordnen lassen. Greene et al. [GKM93] geben einige Lösungsvorschläge, welche aber entweder für dynamische Szenen nicht praktikabel sind oder in einer flachen Einsortierung der Schnittelemente resultieren. Eine Alternative zur binären Vorgehensweise ist der Octree [Gla84], welcher eine Unterteilung einer Zelle in acht gleichgroße Tochterzellen vornimmt.

Bounding Volume Hierarchien bieten eine Lösung für das Problem der Schnittelemente, da sich die Lage der Zelle hier ausschließlich an den Elementen orientiert. Die Hierarchie entsteht durch rekursives Vereinen von n Elementen zu einer umfassenden, in der Regel achsenparallelen Zelle. Der Wert n entspricht der Zahl der Tochterzellen und wird auch als Branching Factor bezeichnet. Das Problem der Bounding Volume Hierarchie beruht auf der Identifikation benachbarter Elemente, da ansonsten extreme Überlappungen zwischen den Zellen entstehen können. Derartige Überlappungen verringern die Effizienz vieler Anwendungen. Demgegenüber resultieren Bounding Volume Hierarchien zumeist in balancierten Bäumen.

Als eines der klassischen Forschungsthemen ermittelt die Sichtbarkeitsbestimmung ausgehend von Szene mit Primitiven eine Menge von sichtbaren Fragmenten, deren Projektion auf die Projektionsebene eines Betrachters von keinem anderen Fragment überlagert oder geschnitten wird. Die Fragmente müssen dabei nicht den ursprünglichen Primitiven entsprechen. Insbesondere ist im Vergleich zu den Primitiven die Zahl der sichtbaren Fragmente deutlich geringer, was die eigentliche Motivation hinter der Sichtbarkeitsbestimmung ausmacht. Ein wichtiger Begriff ist hier die Verdeckungstiefe beziehungsweise die Tiefenkomplexität, welche ein Maß repräsentiert, wieviele Primitive ein gegebenes Primitiv verdeckt. Der Wert ist aber von der Position des Betrachters abhängig. Beispiele für Szenen mit hoher Verdeckungstiefe sind Innenraumarchitekturen und Städtemodelle. Erstere haben vor allem den Begriff der Portale geprägt. Dabei handelt es sich um Öffnungen wie etwa Türrahmen oder Fenster, welche dem Betrachter lediglich einen eingeschränkten Sichtbereich auf benachbarte Räume ermöglichen.

Zu Beginn wurden Hidden-Surface-Removal (HSR) Algorithmen entwickelt, die auch unter der Bezeichnung Visible Surface Determination bekannt sind. Diese Kategorie von Algorithmen bestimmt nicht nur die Menge der sichtbaren Primitive, sondern die exakten Fragmente, welche von einem sichtbaren Primitiv verbleiben. In diesem Zusammenhang ist die exakte Sichtbarkeitsmenge von Bedeutung, welche in Bezug auf einen Betrachter alle zumindest teilweise sichtbaren Primitive einer Szene beinhaltet. HSR Techniken lassen sich in objektpräzise, bildpräzise und hybride Methoden gliedern. Objektpräzise Ansätze [WA77] vergleichen die Primitive einer Szene untereinander, um die sichtbaren Bereiche zu bestimmen. Leider sind objektpräzise Methoden aufgrund der begrenzten Rechengenauigkeit der Computer kaum robust zu implementieren und erfordern zudem einen hohen Rechenaufwand zu Lasten des Hauptprozessors. Bildpräzise Verfahren gehen von der diskreten Auflösung des resultierenden Bildes aus und bestimmen für jeden Pixel das sichtbare Fragment. Als wichtigster Vertreter hat der z-Buffer [Cat74] seinen Weg bis in die moderne Grafik-Hardware gefunden. Hybride Ansätze kombinieren objektpräzise und bildpräzise Methoden und bieten als positive Eigenschaft häufig die Berücksichtigung transparenter Primitive [FKN80].

Im Gegensatz zu HSR versuchen Visibility Culling Verfahren, das Rendern nicht sichtbarer Primitive zu vermeiden und bestimmen hierfür die sichtbare Menge der Primitive. Ein einfaches Beispiel ist Backface Culling, welches vom Betrachter abgewandte Primitive identifiziert [KMGL99]. View Frustum Culling Verfahren berücksichtigen ausschließlich die Primitive innerhalb der Sichtpyramide des Betrachters. Sie machen dabei häufig von Raumunterteilungsbäumen oder Bounding Volume Hierarchien Gebrauch, um ganze Bereiche als unsichtbar zu deklarieren [AM00, Cla76, SC96]. Occlusion Culling Verfahren gelten als ausgabesensitiv beziehungsweise output sensitive, da ihre Laufzeit nicht superlinear zur Anzahl der Primtive einer Szene, sondern proportional zur Anzahl der sichtbaren Primitive ausfällt. Dabei sind zwei weitere Arten der Sichtbarkeitsmenge zu berücksichtigen, nämlich die konservative Sichtbarkeitsmenge und die approximative Sichtbarkeitsmenge. Erstere beinhaltet die exakte Sichtbarkeitsmenge und klassifiziert zusätzlich nicht sichtbare Primitive als sichtbar. Letztere umfasst einige Primitive der exakten Sichtbarkeitsmenge und zusätzlich einige weitere unsichtbare Primitive.

Anstelle der exakten Sichtbarkeitsmenge bestimmen viele Occlusion Culling Verfahren lediglich die konservative oder approximative Sichtbarkeitsmenge und überlassen die genaue

Auflösung des Sichtbarkeitsproblem dem z-Buffer. Aufgrund der heutigen Realisierung von z-Buffer, Backface Culing und View Frustum Culling in Hardware muss der Aufwand zur Bestimmung und Visualisierung der jeweiligen Sichtbarkeitsmenge deutlich geringer sein, als die komplette Szene einfach an die Grafik-Hardware zu übergeben. Um das Ziel der Ausgabesensitivität zu erreichen, werten Occlusion Culling Verfahren drei verschiedene Arten von Kohärenzen aus, nämlich räumliche Kohärenzen, bildbasierte und temporäre Kohärenzen. Erstere entspricht der Auswertung von Raumunterteilungsbäumen. Bildbasierte Kohärenzen berücksichtigen Beziehungen der Primitive im Bildbereich [GKM93, GK94]. Temporäre Kohärenzen gehen von kontinuierlichen Bewegungen des Betrachters aus, weshalb zwischen zwei Frames nur geringe Abweichungen der Sichtbarkeitsmenge zu erwarten sind.

Occludergestützte Occlusion Culling Verfahren bestimmen eine Menge von Primitiven, sogenannten Occluder, welche einen Großteil der in Bezug zu einem Betrachter hinter ihnen liegenden Szene verdecken. In der Regel fällt die resultierende Menge äußerst klein aus. Ein Problem ist die effiziente Selektion der Occluder, wofür häufig vorberechnete Datenbanken verwendet werden. Corg et al. [CT97] beschreiben eine einfache Formel für die dynamische Auswahl der Occluder. Nach der Selektion der Occluder werden die restlichen Primitive der Szene gegen die Occluder getestet, wobei viele Parallelen zu Schattenwurf Verfahren auftreten. Zur Reduktion des entstehenden Aufwands fassen Occluder Fusion Techniken mehrerer Occluder zu einer einzelnen Repräsentation zusammen [BHS98].

Ein weitere Gruppe von Verfahren basiert auf dem Prinzip der Potentially Visible Sets (PVS) [TS91]. Die Szene wird in mehrere Zellen unterteilt und daraufhin für jede Zelle eine Sichtbarkeitsanalyse durchgeführt. Die Ergebnisse werden in der Zelle abgespeichert und mit dem Betreten des Betrachters abgerufen. Aufgrund des hohen Rechenaufwands erfolgt die Analyse zumeist in einem Vorverarbeitungsschritt, weshalb PVS Verfahren nur bedingt für dynamische Szenen geeignet sind. Ein weiterer Nachteil ist der pro Zelle anfallende Speicherbedarf, weshalb PVS Konzepte hauptsächlich in Szenen mit hoher Verdeckungstiefe zum Einsatz kommen. Andererseits bieten sie hinsichtlich derartiger Szenen geringe Laufzeiten.

Leider existieren nur sehr wenige Verfahren, die dynamische Szenen unterstützen. Ein Teil der Verfahren wird als für dynamische Szenen ungeeignet erachtet, weil sie Gebrauch von einem Raumunterteilungsbaum machen. Die vorliegende Arbeit geht aber davon aus, dass eine Lösung für die konsistente Pflege der räumlichen Kohärenzen eines Raumunterteilungsbaumes in Echtzeit existiert. Eine ausgabesensitive Visualisierung generischer, dynamischer 3D Szenen erscheint nur mit Hilfe der gegebenen Grafik-Hardware möglich [ZMHHI97, BMH99]. So bieten alle neueren Grafik-Chips der NVIDIA- oder Ati-Reihe das HP-Flag [HPF] und die NVIDIA-Occlusion-Query an. Das HP-Flag gibt für jedes gerenderte Primitiv ein Signal zurück, ob bei dem Rendervorgang der z-Buffer verändert wurde oder nicht. Die NVIDIA-Occlusion-Query ist eine Erweiterung des HP-Flags und bestimmt nicht nur ein einzelnes Bit, sondern die Anzahl der wirklich gesetzten Pixel eines gerenderten Primitives.

Ähnlich den Polygonnetzen beschreiben auch verteilte Anwendungen einen Graphen, wobei die Teilnehmer der Applikation den Knoten und die Konnektivität der Benutzer den Kanten entsprechen. Die gewählte Topologie hat große Auswirkungen auf den Kommunikationsaufwand, der mit der Synchronisation dynamischer Szenen anfällt. Zwei grundlegende Ansätze sind Peer-to-Peer und Client-Server Systeme. Erstere gehen von einer Gleichstellung der

Teilnehmer aus und erlauben die direkte Kommunikation der Knoten untereinander. Dabei ist zwischen Unicast, Broadcast und Multicast Netzwerken zu unterscheiden. In einem Unicast Netzwerk muss jeder Knoten eine Nachricht eigenständig an alle anderen Teilnehmer versenden. Ein Broadcast Netzwerk verteilt die Nachricht automatisch an sämtliche Knoten, weshalb der Absender nur eine Nachricht verschicken muss. Im Falle der Synchronisation dynamischer Szenen müssen die Empfänger aber immer noch sehr viele Nachrichten verarbeiten. Aus diesem Grund werden die Teilnehmer in einem Multicast Netzwerk nach bestimmten Kriterien in Untergruppen sortiert und eine Nachricht nur an die mit dem Absender assoziierte Gruppe gesendet. Peer-to-Peer Ansätze bieten den Vorteil der gleichmäßigen Auslastung der Knoten, verursachen aber einen hohen Kommunikationsaufwand bei der Synchronisation einer Szene. Innerhalb Client-Server Systeme erfolgt die Kommunikation ausschließlich zwischen Client und Server. Der Server verfügt über eine zentrale Szenenbeschreibung und filtert eingehende Nachrichten vor deren Weiterleitung, weshalb im Vergleich zu Peer-to-Peer Ansätzen ein geringerer Nachrichtenaufwand möglich ist. Allerdings sind die Server zumeist ungleich höher belastet als die Clients. Aus diesem Grund kommen hierarchische Server Konzepte [Fun96b] zum Einsatz, wobei ein Teil der Server ausschließlich auf das Versenden der Nachrichten spezialisiert ist.

Kapitel 5

Wissenschaftliche Einordnung

Während Kapitel 2 die grundlegenden Anforderungen und Ziele der vorliegenden Arbeit formuliert, beschreibt Kapitel 3 nicht nur mögliche Anwendungen und Einsatzgebiete bei einer erfolgreichen Umsetzung dieser Anforderungen, sondern macht zudem deutlich, dass bereits verschiedene Techniken existieren und eingesetzt werden. Somit besteht im Bereich der erstellten Anforderungen eine massive Nachfrage an entsprechenden Technologien. Dies gilt umso mehr, da wie in Kapitel 3 aufgeführt, die verwendeten Techniken entscheidende Mängel haben. In diesem Kapitel sollen nun die wissenschaftlichen Bereiche abgesteckt werden, die von einer Realisierung der Anforderungen betroffen sind. Weiterhin soll geklärt werden, ob und welche Ansätze in den jeweiligen Bereichen bereits existieren, die für eine Umsetzung von Nutzen sind. Dazu wird auf die in Kapitel 4 eingeführten Grundlagen zurückgegriffen. Bei Mängeln oder gar nicht Vorhandensein notwendiger Techniken wird ein entsprechender Bedarf abgeleitet.

5.1 Wissenschaftliche Bereiche

Anforderung 2.1 formuliert die Notwendigkeit einer geeigneten Repräsentation großer, dynamischer und interaktiver 3D Szenen. Wie unter anderem an den Abschnitten 4.1, 4.4, 4.5 und 4.7 zu erkennen ist, handelt es sich bei der Repräsentation von 3D Szenen um einen wichtigen Bereich der Computer Grafik. Der Begriff der Interaktion ist nach Kapitel 2 in die User-Interaktion, die User-Element-Interaktion und die Element-Element-Interaktion zu unterteilen. Die beiden ersten Formen fallen in den Bereich der Mensch-Maschinen-Interaktion, da hier eine Kommunikation zwischen Benutzer und Endgerät erforderlich ist. Die Element-Element-Interaktion tritt üblicherweise infolge dynamischer Vorgänge auf, für deren Beschreibung entweder Animations- oder gar Simulationsvorschriften benötigt werden. Aufgrund dieser kausalen Beziehung werden hier derartige Vorschriften und die Element-Element-Interaktion zum Bereich der Simulation, Animation und Interaktion zusammengefasst. Anforderung 2.2 impliziert einen weiteren wichtigen Bereich der Computer Grafik, nämlich die Visualisierung großer, dynamischer und interaktiver 3D Szenen als solche. Zusätzlich verlangt Anforderung 2.2 die Visualisierung der Szenen auf heterogenen, verteilten

Endgeräten. Hierzu müssen die Informationen der Szene zu den Endgeräten übertragen werden, was zum Bereich der Übertragung überleitet. Da einerseits die begrenzte Bandbreite den Durchsatz an Daten stark einschränkt, andererseits aber mit einer großen 3D Szene eine gewaltige Datenmenge anfällt, werden Algorithmen zur Kodierung und Kompression benötigt. Übertragung, Kodierung und Kompression sind hier zu einem Bereich vereinigt, welcher zusätzlich die Umkehrung, also Dekodierung und Dekomprimierung, mit einschließt. Die folgende Liste führt noch einmal alle relevanten Bereiche auf.

- Szenen Repräsentation
- Mensch-Maschine-Interaktion
- Simulation, Animation und Interaktion
- Visualisierung
- Übertragung, Kodierung und Kompression

In den nächsten Abschnitten werden nun die zuvor ermittelten Bereiche einzeln betrachtet. Ziel ist die Analyse folgender Punkte:

- Welche Technologien werden benötigt?
- Existieren bereits entsprechende Technologien?
- Wenn ja, sind diese Technologien verwertbar?

5.1.1 Szenen Repräsentation

Abschnitt 4.7 erläutert das grundlegende Prinzip der Szenegraphen einschließlich dem damit verbundenen Zustandsmodell. Für eine Repräsentation großer Szenen ist dieses Prinzip alleine jedoch nicht ausreichend. Das hätte nämlich zur Folge, dass alle Elemente der Szene in die Wurzel des Szenegraphen eingeordnet würden. Somit ergäbe sich für die Suche nach einem Element unter Vorgabe der Position eine Komplexität von $O(n)$ und für das Problem der Sichtbarkeitserkennung im schlimmsten Fall eine Komplexität von $O(n^2)$. Eine ausgabesensitive Visualisierung wäre somit nicht möglich. Aus diesem Grund erlauben die meisten Szenegraph-Architekturen eine Integration des Konzepts der Raumunterteilungsbäume beziehungsweise der Bounding Volume Hierarchien (siehe Abschnitt 4.4 und 4.5) mit Hilfe spezieller Knotentypen [1]. Im Falle der Raumunterteilungsbäume ergibt sich dabei das Problem der Schnittelemente, welches für statische Szenen mit den beiden ersten Lösungsvorschlägen von Greene et al. sicherlich lösbar ist. Dies gilt jedoch nicht für dynamische Szenen, zum einen weil eventuell neue Schnittelemente auftreten und zum anderen weil aktuelle Schnittelemente sich wieder eindeutig zuordnen lassen. Letzterer Fall ist insbesondere bei einer Einsortierung der Schnittelemente in alle betroffenen Knoten mit Aufwand verbunden. Ein

[1] Siehe beispielsweise die *Group* Knoten in VRML.

Lösungsvorschlag für das Problem der Schnittelemente stellen die Nine-Area-Trees (NA-Tree) von Chang et al. [CLC03] dar. Allerdings beschränken sie sich auf den zweidimensionalen Fall, weshalb zumindest eine Erweiterung auf dreidimensionale Szenen erforderlich ist. Ihre Idee sieht vor, die Schnittelemente einer gesonderten Sortierung zu unterziehen. Lassen sich die Elemente einer Szene nicht eindeutig einem Quadranten des Quadtrees zuordnen, so wird versucht, eine k-D-Tree Sortierung vorzunehmen und die Elemente entsprechend einem der vier möglichen Halbräume zuzuweisen. Konsequenterweise müssen sich alle Elemente, welche auch diesen Test nicht bestehen, im Zentrum der Zelle schneiden. Unter der Annahme, dass eine derartige Konstellation in normalen 3D Szenen äußerst ungewöhnlich ist und somit nur wenige Elemente betroffen sind, werden diese Elemente direkt der Zelle zugeordnet.

Ein weiteres Problem der Raumunterteilungsbäume ist die konsistente Pflege der räumlichen Kohärenzen zwischen den Elementen einer dynamischen 3D Szene. Auf diesen dynamischen Szenen sind drei Basisoperationen zu definieren:

- Das Einfügen eines Elements in die Szene zur Laufzeit: Diese Situation tritt ein, wenn Elemente zu einem Endgerät übertragen und dort in den Szenegraph eingefügt werden. Sie kann aber auch Folge einer Simulation sein, beispielsweise wenn im Rahmen einer Fußball-Simulation im wahrsten Sinne des Wortes ein neuer Ball ins Spiel kommt.
- Das Entfernen eines Elements aus der Szene zur Laufzeit: Da viele Endgeräte aufgrund ihrer begrenzten Speicherkapazität nicht die gesamte 3D Szene verwalten können, müssen sie bei der Gefahr eines Speicherüberlaufes Informationen der Szene wieder verwerfen. Hierbei ist zu bedenken, dass einige mobile Endgeräte wie etwa PDAs über keine Festplatte verfügen. Analog zum ersten Punkt kann das Entfernen eines Elements auch simulationsbedingt geschehen.
- Die Transformation eines Elements innerhalb der Szene: Hierfür zeichnen sich Animations- und Simulationsvorschriften sowie User-Element-Interaktionen verantwortlich.

Aufgrund der oben genannten Operationen kann es zur einer Verletzung der Kapazität δ einer Zelle kommen, weshalb die Struktur des Raumunterteilungsbaumes aktualisiert werden muss. Eine Aktualisierung des Baumes ist eine kostspielige Angelegenheit, da bei einer Überschreitung von δ der betreffende Knoten v unterteilt oder bei einer Unterschreitung von δ innerhalb des Teilbaumes T_v mit v als Wurzel wieder zusammengefasst werden muss. Dieses Problem ist auch der Grund, warum die Verwendung einer Bounding Volume Hierarchie nicht in Frage kommt. Zum einen ist die vollständige Sortierung aller Elemente einer Szene entlang einer oder mehrerer Hauptachsen für jeden Zeitschritt der Animation oder Simulation zu kostspielig und zum anderen lässt sich hierdurch das Problem der Überlappung lediglich einschränken. Bei der Transformation eines Elements innerhalb einer Bounding Volume Hierarchie könnte die betroffene Zelle entsprechend ausgedehnt werden. Doch wie weit soll eine Zelle maximal dehnbar sein, zumal wenn dadurch das Problem der Überlappung noch verschärft wird? Dieses Problem betrifft übrigens auch die in Abschnitt 4.4 erwähnten Sloppy-n-Arrays. Was passiert, wenn das Element aus der Zelle entfernt und einer anderen Zelle zugewiesen werden muss? In diesem Fall könnte eine Verletzung des Branching Factors auftreten, so dass eine massive Umstrukturierung der Hierarchie erforderlich ist. Da

die Zellen im Gegensatz zu den Raumunterteilungsbäumen nicht eindeutig einer bestimmten Region der Szene zugeordnet sind, fallen derartige Umstrukturierungen äußerst teuer aus.

An dieser Stelle bleibt somit festzuhalten, dass ein neues Konzept für die konsistente räumliche Repräsentation dynamischer Szenen benötigt wird. Für das Problem der Schnittelemente kann der Vorschlag von Chang et al. aufgegriffen werden, wobei allerdings eine Erweiterung auf den dreidimensionalen Raum anfällt. Zusätzlich besteht nach Kapitel 2 der Anspruch, das gewünschte Konzept auch für *out-of-core* Rendering auszulegen, d.h. für die konsistente räumliche Repräsentation dynamischer Szenen, welche aufgrund ihrer Größe teilweise auf externe Geräte ausgelagert sind. Zum Thema des out-of-core Rendering gibt es Ansätze von Klein et al. [KKF+02] und Varadhan et al. [VM02], die aber nicht die Möglichkeit dynamischer Szenen vorsehen. Während Klein et al. einen normalen Octree verwenden, machen Varadhan et al. keine näheren Angaben zu ihrer Raumunterteilungsstruktur. Erikson et al. [EMBI01] beschreiben die Visualisierung großer, dynamischer Szenen. Sie verwenden eine Bounding Volume Hierarchie zur räumlichen Repräsentation der Szenen, gehen jedoch nicht auf eine out-of-core Behandlung der Datenstruktur ein. Nach jedem Aus- und Einfügen eines Elements muss die Hierarchie neu aufgebaut werden. Ansonsten versuchen Erikson et al. bei geringen Bewegungen eines Elements die betroffene Zelle wie bereits oben angesprochen auszudehnen. Funkhouser at al. beschreiben ein Speichermanagement für das Aus- und Einlagern grafischer Objekte [FS93, Fun96a]. Sie legen in ihrer Arbeit den Schwerpunkt auf räumlich unterteilte, statische Architekturmodelle. Weitere Ansätze und Vorschläge stammen von Avila et al. [AS97], Chim et al. [CGL+98], Shou et al. [SCH+01] und Correa et al. [CKS02], welche sich aber ebenfalls auf statische Szenen beziehen.

Während der bisherige Bereich des Abschnitts die räumliche Sortierung der Elemente einer Szene behandelt, befasst sich der folgende Teil mit der Repräsentation der Elemente an sich. Abschnitt 4.2 beschreibt hierzu den objektorientierten Ansatz der Animationslemente. Da nach Anforderung 2.2 eine Visualisierung auf heterogenen Endgeräten mit unterschiedlichen Leistungsmerkmalen möglich sein soll, ist wie bereits in Abschnitt 4.2 angedeutet ein flexibleres Konzept als der klassische objektorientierte Ansatz vonnöten. Ansonsten würde das Animationselement mit einer Vielzahl von Visualisierungsmethoden für das jeweilige Betriebssystem oder die jeweilige Grafik-Bibliothek überladen, deren Nutzung auf einem Endgerät aber schließlich nur auf einige wenige beschränkt wäre. Hier kann der komponentenbasierte Ansatz weiterhelfen, der als Erweiterung zur Objektorientierung noch die Beschreibung eines sogenannten *Frameworks* fordert. Ein Framework definiert das Interaktionsmodell der *Komponenten*. Bei diesen Komponenten handelt es sich um Objekte, deren Schnittstellen allerdings durch das vorgegebene Framework normiert sind. Hierdurch ist es möglich, eine Funktionalität über den Austausch zweier Komponenten zu verändern, ohne dabei Auswirkungen auf die restliche Architektur zu verspüren. Somit wäre es möglich, die Methoden zur Visualisierung eines Elements in einer systemspezifischen Komponente zu kapseln. Betritt ein Element das System, so wird die dort vorhandene Komponente verwendet. Dies impliziert jedoch, dass sowohl für die Animationselemente [2] als auch für die Visualisierungskomponente ein entsprechendes Framework spezifiziert werden muss. Eine Weiterführung des komponentenbasierten Ansatzes stellen Agenten-Technologien dar, die zusätzlich noch eine stärkere Normierung der Kommunikationsschnittstelle sowie Schemata für Aufgabenbeschreibungen

[2] In diesem Sinne wäre auch eine Bezeichnung als Animationskomponente sinnvoll.

vornehmen. In dieser Arbeit ist der Begriff der Agenten nicht im klassischen Sinne zu verstehen (siehe [MWJ97]). Vielmehr stellen Agenten hier Komponenten dar, die über eine entsprechende Aufgabenbeschreibung Aktionen ausführen können. Bei diesen Beschreibungen kann es sich beispielsweise um Animations- oder Simulationsvorschriften handeln. Im folgenden wird daher der Begriff der Animationselemente zum Begriff der Animationsagenten erweitert. Eine äquivalente Bezeichnung existiert bereits in [Dör01], allerdings werden dort die Methoden zur Visualisierung auf traditionelle Art und Weise durch den Agenten selbst offeriert.

Für eine Visualisierung auf heterogenen Endgeräten ist eine adaptive Repräsentation der eigentlichen visuellen Informationen eines Animationsagenten erforderlich. Hierzu sind LOD-Konzepte zur automatischen Generierung von Detailstufen geeignet, die, wie in Abschnitt 4.3 unschwer zu erkennen, bereits in einer Vielzahl verschiedener Ansätze existieren. Nach Kapitel 2 sind jedoch einige Aspekte zu berücksichtigen, welche die Menge der im Sinne dieser Arbeit verwendbaren Vorgehensweisen stark einschränkt. Für eine effiziente Visualisierung in Echtzeit ist in jedem Fall eine Verwendung der vorhandenen Grafik-Hardware unabdingbar. Aktuelle Grafik-Chips unterstützen jedoch beinahe ausschließlich die Ausgabe von Polygonnetzen, in der Regel sogar nur von Dreiecksnetzen. PDAs bieten derzeit noch keinen Grafik-Support durch die Hardware, dennoch haben die für PDAs einsetzbaren Grafik-Bibliotheken wie etwa [KLI] identische Einschränkungen. Vermutlich wird es bald erste Grafik-Chips für wirklich mobile Geräte geben. Die Wahrscheinlichkeit, dass deren Funktionalität grundlegend anders ausfällt als bei der aktuellen Hardware, ist aber äußerst gering. Als Konsequenz müssen mit Hilfe von impliziten Funktionen, parametrischen Funktionen, Multiresolution Analysis oder Unterteilungsflächen erstellte Modelle in Polygonnetze mit den Detailstufen entsprechenden Auflösungen transformiert werden. Dies ist aber derzeit noch nicht einmal von Standard PCs, geschweige denn von mobilen Endgeräten, für mehrere Elemente und Detailstufen in Echtzeit leistbar. Somit scheidet die Möglichkeit aus, die Kompaktheit der genannten Repräsentationen für eine effiziente Übertragung zu nutzen, um dann vor Ort die erforderlichen Umwandlungen in Polygonnetze vorzunehmen. Allerdings wäre es nach wie vor denkbar, derartige Repräsentationen auf leistungsfähigeren Endgeräten für eine Vorberechnung der benötigten Polygonnetze einzusetzen, um diese dann an die schwächeren Endgeräte zu transferieren. Leider gerät dieser Ansatz in Konflikt mit der in Kapitel 2 geforderten Adaptivität der Daten. Da die Leistungsmerkmale der Endgeräte aufgrund der Vielzahl möglicher Hardware-Kombinationen kaum zu diskretisieren sind, wird eine Repräsentation benötigt, welche sehr schnell nahezu stufenlose Auflösungen extrahieren kann. Selbst der geringste Detailgrad, der sich auf einem Standard PC mühelos prozessieren lässt, kann schon die Fähigkeiten eines PDAs bei weitem übersteigen. Im Falle einer Vorberechnung würde dies für ein Polygonnetz aus n Polygonen eine fast identische Zahl von Auflösungen und somit bei größeren Modellen einen gigantischen Speicherbedarf implizieren. Eine Berechnung der Polygonnetze zur Laufzeit ist bei gleichzeitiger Anfrage verschiedener Endgeräte selbst mit einem sehr leistungsfähigem Endgerät nicht über die aufgeführten Techniken in einer akzeptablen Reaktionszeit realisierbar. Somit bleiben zur Repräsentation und zur Generierung von Detailstufen lediglich Ansätze übrig, die direkt auf Polygonnetzen arbeiten. Da sowohl die entsprechenden Verfahren als auch die aktuelle Grafik-Hardware nahezu ausschließlich von Dreiecksnetzen Gebrauch machen, wird im folgenden von dieser spezialisierten Form der Polygonnetze ausgegangen.

Aufgrund der exakten Adaptivität der Daten ist das Ziel die Option einer genauen Vorgabe der Anzahl an Dreiecken n, die an ein Endgerät übertragen werden soll. Damit scheiden auch Resampling Methoden aus, da hier die Anzahl der resultierenden Dreiecke nicht vorhersagbar ist (siehe Abschnitt 4.3.2). Trotz der eventuell geringeren Dreieckszahl einer Detailstufe gegenüber dem Original, soll nicht nur eine optisch ansprechende Visualisierung möglich sein. Vielmehr soll der Benutzer bereits nach k transferierten Dreiecken mit $k < n$ ein Modell als solches identifizieren können, d.h. gefordert ist eine möglichst gute Annäherung der Modelle M_i mit $i = 1, \ldots, n$ an das Original [3]. Hierzu sind progressive Verfahren zur Geometriereduktion geeignet, weil diese erst ein sehr einfaches Basismesh M^0 generieren und selbiges dann schrittweise verfeinern (siehe Abschnitt 4.3.1). Eine grundlegende Anforderung ist dabei, dass zumindest das Basismesh auf jedem Endgerät in Echtzeit zu visualisieren ist. Die Restriktion auf progressive Verfahren zur Geometriereduktion engt den Kreis verwertbarer Technologien bereits sehr stark ein. Übrig bleiben unter anderem die Ansätze von Hoppe [Hop96], Popovic et al. [PH97] und Schröder [Sch97].

Da die einzelnen Detailstufen nicht nur in Bezug auf die Geometrie eine gute Annäherung auf das Original bieten sollen, ist auch eine Berücksichtigung der Zusatzattribute eines Modells gefordert. Zu diesen Attributen sind im Rahmen dieser Arbeit Normalen, Farben und Texturkoordinaten zu zählen. Es soll außerdem sowohl eine *per vertex* als auch eine *per facet* Definition der Attribute möglich sein. Schröder geht nicht auf eventuelle Zusatzattribute ein. Garland et al. [GH98] beschreiben eine Variante ihres QEM-Verfahrens, die speziell auf die Berücksichtigung von Farben und Texturen ausgelegt aber leider nicht progressiv ist.

Eine Gemeinsamkeit der somit verbleibenden Ansätze von Hoppe und Popovic et al. liegt im hohen Rechen- und Zeitaufwand für das Erstellen der progressiven Daten. Hoppes Intention ist in erster Linie die exakte Annäherung an das Original, wozu er ein Optimierungsproblem löst (siehe Abschnitt 4.3.1). Der Ansatz von Popovic mit Hoppe als Co-Autor stellt letztlich ein Derivat von Hoppes Verfahren dar, insofern gibt es hier keine Verbesserung der Laufzeit. Die Manipulation eines Modells während der Laufzeit beispielsweise durch das Verformen des Dreiecksnetzes ist daher über diese beiden Verfahren nur mit sehr langen Wartezeiten möglich. Hier besteht also der Bedarf, ein schnelles progressives Verfahren zu entwickeln, welches eventuelle Zusatzattribute berücksichtigt. Schnell bedeutet dabei nicht nur ein effizientes Generieren der progressiven Informationen, sondern insbesondere das sehr einfache und kostengünstige Verfeinern der Dreiecksnetze auf den jeweiligen Endgeräten.

Neben den zuvor genannten Eigenschaften gibt es aber noch weitere Anforderungen an die zu entwickelnde Technik, die derzeit nicht von existierenden Verfahren unterstützt werden. Informationen, die ein Endgerät erreichen, sollen dort sofort in Echtzeit visualisierbar sein, d.h. es soll nicht bis zur Vervollständigung einer bestimmten Detailstufe gewartet werden. Weiterhin soll sowohl die Verfeinerung eines Dreiecksnetzes als auch dessen Visualisierung mit Hilfe der gleichen Datenstruktur möglich sein. Hier besteht eine Abgrenzung zu sämtlichen anderen Verfahren, da diese vor einer Visualisierung erst eine aufwändige Konvertierung der Verfeinerungsdatenstrukturen zu einer effizient renderbaren Datenstruktur vornehmen müssen. Zusatzattribute sollen nach der Umwandlung eines Modells in das progressive Daten-

[3] Nicht zu verwechseln mit der Hoppe Notation (siehe Abschnitt 4.3.1). Der Parameter i ist hier in Bezug auf die Anzahl der Dreiecke des Modells M_i zu sehen.

format in getrennten progressiven Datenströmen [4] vorliegen. Dafür gibt es folgende Gründe:

- Nicht jedes Endgerät unterstützt alle Zusatzattribute, so ist beispielsweise eine texturierte Visualisierung auf PDAs nur bedingt möglich. Durch eine Trennung der Datenströme können die unterstützten Datentypen einfach selektiert werden. Die unnötige Übertragung eines Datentyps infolge eines alles beinhaltenden Formats wird vermieden.
- Für jeden Datenstrom kann eine spezielle und effiziente Technik zur Kodierung und Kompression verwendet werden. Gleiches gilt auch für die Dekodierung und Dekomprimierung der Daten.
- Die einzelnen Datenströme können mehrfach genutzt werden. So ist es beispielsweise unnötig, für zwei Modelle, die sich lediglich in der Farbe unterscheiden, jeweils vollständige progressive Repräsentationen zu speichern. Vielmehr soll es eine vollständige Repräsentation und eine zusätzliche Farben-Repräsentation geben. Dies führt nicht nur zur Reduktion des Speicheraufwandes, sondern auch des Übertragungsaufwandes. Ist das rote Modell M bereits komplett übertragen, so muss nicht auch noch das schwarze Modell M vollständig transferiert werden, sondern der schwarze Farbenstrom reicht völlig aus.

Abschließend ergeben sich somit für das benötigte Verfahren die hier aufgelisteten Anforderungen:

1. Progressiv
2. Berücksichtigung von Zusatzattributen (Normalen, Farben, Texturkoordinaten)
3. Geringe Laufzeit sowohl beim Generieren als auch beim Verfeinern
4. Verfeinerung bei gleichzeitiger Visualisierung in Echtzeit
5. Trennung der Datenströme

5.1.2 Mensch-Maschine-Interaktion

Der Schwerpunkt der vorliegenden Arbeit liegt nicht im Bereich der Mensch-Maschinen-Interaktion. Hierzu gibt es einschlägige Literatur, unter anderem aus dem Bereich der Software-Ergonomie. Kapitel 2 fordert allerdings eine Gleichberechtigung mobiler Endgeräte, d.h. die Benutzer derartiger Geräte sollen nicht durch die Beschränkung der Eingabemöglichkeiten benachteiligt werden. Für Laptops existieren derzeit zu einem Standard PC vergleichbare Eingabegeräte, unter anderem Tastatur, Maus, Tracking Ball und Joystick. Anders sieht es dagegen bei PDAs aus, deren einzige direkte Eingabemöglichkeit im PDA-Stift besteht. Hier

[4] D.h. ein Datenstrom für Normalen, ein Datenstrom für Texturen, usw. ...

Abbildung 5.1: *Das Spiel Ultima Underworld bietet eine PDA taugliche Navigationsschnittstelle und ist mittlerweile auch für den Pocket PC erhältlich (Screenshot von Ultima Underworld).*

gibt es also zumindest den Bedarf für ein geeignetes Interface zur User-Interaktion beziehungsweise Navigation und für ein Interface zur Manipulation der Szene. Letzterer Punkt ist dabei in Kombination zu den in Abschnitt 5.1.1 eingeführten Basisoperationen auf dynamischen Szenen zu sehen, weshalb es einfache Eingaben zum Aus- und Einfügen sowie zur Transformation eines Elements geben muss. Für ein Interface zur Navigation innerhalb einer 3D Szene existieren bereits PDA-taugliche Ansätze, die lediglich zu erweitern sind. Ein Beispiel ist eines der ersten PC-Spiele mit echter 3D Grafik überhaupt, nämlich *Ultima Underworld* (siehe Abbildung 5.1) [5].

5.1.3 Simulation, Animation und Interaktion

Aufgrund der Unterstützung dynamischer 3D Szenen ist es erforderlich, für Simulationen und Animationen eine Schnittstelle zur Beschreibung dynamischer Vorgänge anzubieten. Zur Erzeugung von Animationen existiert bereits ein ganze Reihe verschiedener Techniken, beispielsweise Interpolation von *Key Frames* [Ree81], *Motion Capturing*[MG01] oder inverse Kinematik [DVS01]. Ziel dieser Arbeit ist nicht die Entwicklung einer neuen Animationstechnik. Vielmehr soll es möglich sein, Animationen, die mit Hilfe der erwähnten Technologien erstellt wurden, auf den jeweiligen Endgeräten anzuzeigen. Wie in Kapitel 3 beschrieben wird hiervon schon in verschiedenen Anwendungsgebieten Gebrauch gemacht, unter anderem in Virtual Chatrooms, Produktwerbungen und 3D Spielen. Letztere gehen davon aus,

[5] Dieses Spiel ist mittlerweile auch für den Pocket PC erhältlich.

dass die benötigten Informationen bereits auf dem jeweiligen Endgerät vorhanden sind, um dann nur noch die Position und die aktuelle Framenummer der Mitspieler untereinander auszutauschen [6]. Faure et al. [FFH+99] bezeichnen diese Vorgehensweise als *Online Animation*. Virtual Chatrooms haben einen ähnlichen Ansatz, müssen aber bei einer Verteilung über das Internet zumindest erst einmal vorhandene Animationen übertragen. Produktwerbungen beschränken sich in der Regel auf äußerst einfache Transformationen wie Skalierung und Rotation. Im Gegensatz zu den beiden anderen Anwendungsbeispielen benötigen sie keine Synchronisation zwischen den einzelnen Endgeräten. Stattdessen treffen die Benutzer der Endgeräte mit dem Starten, Beenden oder Wiederholen einer Animation unabhängig voneinander ihre eigene Zeitvorgabe. Faure et al. deklarieren dies als *Offline Animation*.

Die Notwendigkeit sich im Falle der 3D Spiele oder der Virtual Chatrooms auf einige wenige Informationen wie etwa die Position zu beschränken, ist durch die wachsenden Latenzzeiten bei der Übertragung größerer Datenmengen gegeben. Hohe Latenzzeiten erschweren die Synchronisation verteilter dynamischer Vorgänge, da es hierdurch zu längeren Blockaden der beteiligten Endgeräte kommen kann. Aus diesem Grund gibt es verschiedene Ansätze, deren Intention die Minimierung der Netzwerkbelastung innerhalb kollaborativer Umgebungen ist [CET99, MZP+94]. Die Synchronisation verteilter, dynamischer Szenen stellt ein wichtiges Kriterium für die Wahl zwischen einem zentralen Client-Server-System und einem dezentralen Peer-to-Peer-System dar (siehe Abschnitt 4.8). Zentrale Ansätze sind in der Regel einfacher zu synchronisieren, da hier zumindest eine konsistente Repräsentation der 3D Szene besteht. Eventuelle Manipulationen müssen erst dem Server gemeldet werden, welcher die Änderungen an seiner Repräsentation vornimmt und dann an alle registrierten Endgeräte weiterleitet. Hierdurch ist es auch leichter, Element-Element-Interaktionen auszuwerten, eine Aufgabe, die innerhalb dieser Arbeit der Simulation obliegt. Für eine klassische Form der Element-Element-Interaktion, nämlich der Kollisionserkennung, existiert eine Vielzahl verschiedener Ansätze, unter anderem von [KHM+98].

An dieser Stelle bleibt somit festzuhalten, dass es einen Bedarf für eine Schnittstelle zur Beschreibung dynamischer Vorgänge gibt. Diese Schnittstelle wird von einer der Szene zugrunde liegenden Animation oder Simulation angesprochen. Etwaige Element-Element-Interaktionen sind von der Simulation aufzulösen, da nur diese die jeweiligen Ausprägungen der Interaktion kennt. Techniken zur Übertragung und Synchronisation vorberechneter Animationen werden bereits von derzeitigen Applikationen benutzt und können von dort übernommen werden. Jedoch gibt es eine weitere Anforderung, die von anderen Ansätzen nur bedingt erfüllt wird, nämlich die Skalierbarkeit der zu übertragenden Animationen: Zum einen sollen Animationen, welche der Betrachter einer Szene niemals zu Gesicht bekommt, nicht übertragen werden. Zum anderen sollen Animationen, die beim Laden einer Szene im Sichtbereich des Betrachters aktiv sind, priorisiert übertragen werden. Capin et al. [CJE+98] bieten hierzu eine spezialisierte Lösung für das Transferieren menschlicher Körper. Sie verwenden einen Standardkörper, der beim ersten Zugriff eines Benutzers mitsamt seiner Freiheitsgrade auf das Endgerät geladen und dann durch den konkreten Charakter verfeinert wird. Bei folgen-

[6] Aktuelle 3D Spiele verwenden nicht nur vorberechnete Animationen, sondern sind auch zur Laufzeit in der Lage, mittels inverser Kinematik Skelettanimationen zu bestimmen. Derartige Kalkulationen müssen nur von einem Endgerät vorgenommen und dann in Form einfacher Positions- oder Transformationsangaben an die anderen Teilnehmer übertragen werden.

den Zugriffen wird der nun bereits vorhandene Standardkörper analog eingesetzt.

5.1.4 Visualisierung

Wie in Abschnitt 5.1.1 beschrieben soll mit Hilfe des Konzepts der Animationsagenten die Visualisierung eines Animationselements auf heterogenen Endgeräten realisiert werden. Hierdurch soll es nicht nur möglich sein, identische Visualisierungen auf differierenden Systemen umzusetzen, sondern zusätzlich soll auch die Option beliebiger Arten der Visualisierung bestehen. So kann beispielsweise aufgrund der hohen Auslastung eines Endgerätes der Wunsch aufkommen, eine Vereinfachung der Visualisierung zu erreichen, die über das Konzept der Level-of-Detail hinausgeht. Eine entsprechende Lösung könnte etwa in einem *Level-of-Abstraction* Konzept bestehen [Pin88], welches anstelle der vollständigen visuellen Informationen lediglich die Bounding Boxes der einzelnen Animationsagenten anzeigt.

So flexibel das Konzept der Animationsagenten auch sein mag, letztlich ist seine visuelle Information auf die Beschreibung eines einzelnen, wenn auch möglicherweise komplexen Elements der Szene restriktiert. Große 3D Szenen können jedoch nach Kapitel 2 mehrere Millionen derartiger Elemente beinhalten. Da bei solchen Mengen an Daten selbst die neusten Grafik-Chips überfordert sind, muss eine ausgabesensitive Visualisierung das Ziel sein (siehe Abschnitt 4.6). Occlusion Culling Verfahren stellen hierzu ein geeignetes Mittel dar, allerdings beschränken sich viele Verfahren auf bestimmte Szenarien wie etwa Landschaftsmodelle [Ste97], Stadtmodelle [WS99, WWS00] oder Innenarchitekturen [TS91, LG95a, Fun96a]. Die Vielfalt möglicher Anwendungen (siehe Kapitel 3) erfordert aber einen generischen Ansatz, der im Rahmen beliebiger Szenarien zum Einsatz kommen kann. Aufgrund der Berücksichtigung dynamischer Szenen scheiden PVS-basierte Techniken aus, da sie aufwändige Vorberechnungen für die interaktive Navigation durch eine Szene benötigen. Ein weiterer Kritikpunkt ist der enorme Speicherbedarf zum Verwalten der ermittelten Sichtbarkeitsmengen. Verfahren, welche zuerst eine Occluder-Datenbank [ZMHHI97, BHS98] ermitteln, sind in dynamischen Szenarien ebenfalls nur bedingt einsetzbar. Das in Abschnitt 5.1.1 beschriebene Ein- und Ausfügen von Elementen würde neben der Verwaltung der Szene selbst eine permanente Pflege der Datenbank erfordern. Diese könnte durch eine für jedes Element vorbestimmte Menge potentieller Occluder erleichtert werden. Allerdings würde eine derartige Vorgehensweise wiederum eine zusätzliche Belastung bei der Übertragung eines Elements an die Endgeräte implizieren.

Unter der Voraussetzung, dass eine konsistente Pflege der räumlichen Repräsentation einer Szene in Echtzeit möglich ist, existieren durchaus Techniken [GKM93, BMH99], welche ohne die oben genannten Vorverarbeitungsschritte auskommen und insofern für dynamische Szenen geeignet sind. Greene et al. verwenden einen Octree für die räumliche Sortierung einer Szene, dessen Zellen sie beginnend mit der Wurzel während der Visualisierung in einer in Bezug auf den Betrachter *front-to-back* Sequenz rekursiv traversieren. Jede Zelle wird zunächst gegen die Sichtpyramide des Betrachters geclippt. Befindet sich die Zelle vollständig außerhalb der Sichtpyramide, dann endet die Rekursion an dieser Stelle. Ansonsten wird die Zelle gegen eine hierarchische z-Pyramide getestet. Scheitert auch dieser Test, d.h. die Zelle ist zumindest teilweise sichtbar, dann müssen alle Elemente innerhalb der Zelle gerendert

und eventuelle Unterzellen in gleicher Weise wie die Zelle überprüft werden. Das Erstellen der z-Pyramide ist ein aufwändiges und speicherintensives Unterfangen. Sie repräsentiert einen Quadtree, dessen höchster Level dem normalen z-Buffer entspricht. Von jeweils vier benachbarten Pixel eines Levels wird der höchste z-Wert bestimmt und in das nächst niedrigere Level eingetragen. Die Anzahl der z-Werte n_l eines Levels l ist also $n_l = \frac{n_{l+1}}{4}$. Die Wurzel des Quadtrees beinhaltet den maximalen z-Wert [7]. Jedesmal wenn das Rendern eines Primitivs Änderungen im z-Buffer verursacht, erfordert dies ein Propagieren der neuen Werte durch die z-Pyramide. Im Gegenzug ermöglicht die z-Pyramide allerdings schnelle Sichtbarkeitstest: Ist beispielsweise der kleinste z-Wert einer Zelle größer als der Wert in der Wurzel des Quadtrees, so ist die Zelle unsichtbar. Ansonsten wird überprüft, in welchem Quadranten die Abbildung der Zelle im Bildbereich liegt und ein identischer Test gegen den z-Wert durchgeführt, der im nächsten Level der Pyramide den betreffenden Quadranten repräsentiert. Dieser Vorgang wird rekursiv fortgeführt. Lässt sich die Zelle nicht eindeutig einem Quadranten zuordnen, so erfolgt ein entsprechendes Clipping. Greene et al. argumentieren gegen den Punkt des hohen Rechen- und Speicheraufwands der z-Pyramide mit deren Option zur eleganten Umsetzung in Hardware. Dennoch konnte sich hier das Verfahren gegen den einfacheren z-Buffer nicht durchsetzen. Bartz et al. haben durchaus einen ähnlichen Ansatz, verwenden aber anstelle der z-Pyramide einen wesentlich kostengünstigeren Test. Sie rendern die Primitive einer virtuellen Zelle (siehe Abschnitt 4.6) in den z-Buffer und analysieren diesen darauf auf eventuelle Änderungen. Eine Manipulation des z-Buffers ist gleichbedeutend mit der Sichtbarkeit des Primitivs und damit auch der Zelle. Ein Problem des Verfahrens ist nicht nur der Aufwand zum Analysieren des z-Buffers, sondern auch dessen erforderliche Restauration nach möglichen Änderungen. Analog zu Greene et al. hängt der Aufwand dabei stark von der gewählten Bildschirmauflösung ab. Die genannten Probleme ergeben sich jedoch nicht, sofern eine Unterstützung durch das HP-Flag besteht. In diesem Fall kann der Ansatz von Bartz et al. als das derzeit vielleicht schnellste Verfahren für generische Szenen angesehen werden und ist daher bei einer Visualisierung großer Szenen nicht zu vernachlässigen.

Leider sind sowohl das HP-Flag als auch die NVIDIA-Occlusion-Query Bestandteil der OpenGL Erweiterungen. Sie gehören also nicht zum Standard und werden daher von einer Vielzahl älterer aber immer noch weit verbreiteter Grafik-Chips nicht unterstützt. Gleiches gilt für die PDAs, die zwar über OpenGL-Bibliotheken aber eben nicht über die entsprechenden Erweiterungen verfügen. Insofern besteht im Rahmen dieser Arbeit das Bedürfnis, ein Occlusion Culling Verfahren zu entwickeln, welches auch ohne Unterstützung des HP-Flag und der NVIDIA-Occlusion-Query eine Echtzeitvisualisierung dynamischer Szenen ermöglicht. Eine weitere Abgrenzung gegenüber existierenden Techniken beruht auf der Anforderung, die Präzision und den Rechenaufwand gemäß der Adaptivität der Algorithmen (siehe Abschnitt 2) den Leistungsmerkmalen eines Endgerätes anpassen zu können. Somit ergeben sich folgende Kriterien für die zu entwickelnde Technik:

- Ausgabesensitiv

[7] Im Zweifelsfall entspricht dieser Wert der Sichtweite, da die z-Pyramide zuvor mit dieser initialisiert werden muss. Zellen und Elemente sind analog zum z-Buffer Verfahren in das Koordinatensystem des Betrachters zu transformieren.

- Unterstützung dynamischer Szenen in Echtzeit
- Keine Verwendung von OpenGL Erweiterungen
- Adaption von Präzision und Aufwand an die Leistungsmerkmale eines Endgerätes

Da der Ansatz von Bartz et al. bei Vorhandensein der Erweiterungen schwer an Geschwindigkeit und Einfachheit zu überbieten ist, soll analog zum Konzept der Animationsagenten ein Framework für das Austauschen beliebiger Sichtbarkeitsverfahren entwickelt werden.

5.1.5 Übertragung, Kodierung und Compression

Nach Kapitel 2 ist die geeignete Selektion der Daten ein wichtiges Mittel, um eine Adaptivität der Daten zu erreichen. Grundsätzlich sollten diejenigen Informationen, welche im Interesse des Benutzers liegen, mit einer höheren Priorität zu dem betreffenden Endgerät übertragen werden als solche außerhalb seines Interesses. Hierdurch wird nicht einfach nur die Menge an Daten reduziert, sondern vor allem auch die Reaktionszeit bis der Benutzer ein Feedback erfährt. Dabei stellt sich die Frage, wie das Interesse eines Benutzers zu spezifizieren ist. Da die Intention dieser Arbeit letztlich in der Visualisierung von 3D Szenen besteht, sollen im folgenden ausschließlich visuelle und entfernungsspezifische Kriterien maßgebend sein. In diesem Zusammenhang ist der Begriff der *Area-of-Interest* von Bedeutung [HS98], welche um den zentralen Standort des Benutzers innerhalb der Szene eine Zone des Interesses definiert. Hier ist also der Abstand eines Elements zum Betrachter das ausschlaggebende Kriterium. Bedauerlicherweise beschreiben Hesina et al. nicht, wie sie die entsprechenden Elemente identifizieren. Dieses Problem ist nämlich einer Suche unter Vorgabe einer Position beziehungsweise eines Gebietes in der räumlichen Repräsentation der Szene gleichzusetzen, d.h. es ist kein triviales Problem. Dies gilt umso mehr, weil mehrere Betrachter gleichzeitig durch die Szene navigieren können. Eine Lösung könnten die in Abschnitt 4.6 eingeführten Occlusion Culling Verfahren sein, welche außerdem auch ein bedeutend präziseres Kriterium wären [8]. Cohen-Or et al. [COZ98, COFHZ98a] beschreiben einen entsprechenden Vorschlag für die Übertragung einer Szene. Allerdings sind nur PVS-basierte Verfahren schnell genug, um die sichtbaren Elemente in Bezug auf mehrere Betrachter in Echtzeit zu bestimmen. Die beschränkte Einsatzmöglichkeit PVS-basierter Verfahren wurde bereits in Abschnitt 5.1.4 diskutiert. Der Ansatz von Bartz et al. [BMH99] ist für eine gleichzeitige Bestimmung mehrerer Sichtbarkeitsmengen zu teuer und erfordert zudem eine geeignete Hardware-Unterstützung. Hier kämen somit nur sehr teure grafikfähige Superrechner, wie etwa von SGI offeriert, in Frage. Teler et al. [TL01] gehen ebenfalls nicht auf die eigentliche Identifizierung der Elemente ein, sondern konzentrieren sich auf die Bestimmung der zu erwartenden Betrachterpositionen im Rahmen einer Navigation des Benutzers. Derartige Aussagen können wie auch bei Hesina et al. für ein *Prefetching* von Elementen verwendet werden, d.h. Elemente, die sich momentan noch außerhalb der Area-of-Interest befinden, aber aufgrund der Navigation des Benutzers bald in sein Blickfeld geraten könnten, werden präventiv übertragen. Hieraus lässt sich eine zumindest dreistufige Priorisierung der Elemente ableiten:

[8] Hesina et al. definieren die Area-of-Interest als einen Kreis mit dem Betrachter im Zentrum.

- Elemente, die sich im Sichtbereich des Betrachters befinden, müssen übertragen werden.
- Elemente, die innerhalb einer bestimmten Zeitspanne in den Sichtbereich des Betrachters geraten könnten, sollten übertragen werden.
- Elemente, für welche die beiden ersten Punkte nicht gelten, brauchen vorerst nicht berücksichtigt zu werden.

Da auch andere Ansätze keine Angaben zur Identifizierung von Areas-of-Interest machen, ist an dieser Stelle der Bedarf für ein neues Area-of-Interest Konzept festzuhalten. Der auf diesem Konzept basierende Algorithmus zur Identifizierung betroffener Elemente soll die folgenden Eigenschaften haben:

- Unterstützung mehrerer Benutzer in Echtzeit
- Unterstützung dynamischer Szenen
- Priorisierung der Elemente

Aufgrund der begrenzten Bandbreiten ist es erforderlich, die zu übertragenden Datenmengen zu kodieren und gegebenenfalls zu komprimieren. Ein erster Ansatz ist hierfür die in Abschnitt 5.1.1 geforderte Trennung der Datenströme bei der Erzeugung der progressiven Informationen. Somit können die für jeden Datentyp optimalen Verfahren zur Kodierung und Komprimierung eingesetzt werden. Verfahren dieser Art existieren schon in vielfältiger Form, beispielsweise der *Edgebreaker* Algorithmus [Ros99] für die Kodierung von Polygonnetzen oder das JPEG2000 Format [JPE] zur progressiven Übertragung von Texturen. Ziel dieser Arbeit ist hier lediglich ein flexibles Konzept, um die vorhandenen Techniken auch einsetzen zu können.

5.2 Existierende Ansätze

Die vorangehenden Abschnitte identifizieren die wissenschaftlichen Bereiche, die von der vorliegenden Arbeit betroffen sind. Weiterhin analysieren sie für jeden dieser Bereiche, welche Techniken für die Umsetzung der in Kapitel 2 formulierten Anforderungen benötigt werden, welche Lösungen es bereits gibt und inwiefern letztere verwendet werden können. Die dortigen Ausführungen sind also sehr spezifisch auf die einzelnen Technologien ausgelegt. Es existieren jedoch auch komplexere Konzepte, deren Intention zumindest ansatzweise ähnlich zu den in Abschnitt 2 aufgeführten Zielen ist. Da diese Konzepte von den derzeit vorhandenen Techniken Gebrauch machen, unterliegen sie konsequenterweise den entsprechenden Einschränkungen. In den folgenden Abschnitten werden nun einige bedeutende Konzepte vorgestellt und diskutiert.

5.2.1 Distributed Interactive Virtual Environment

Obschon der Ursprung des *Distributed Interactive Virtual Environments* (DIVE) [CH93a, CH93b, Hag96, FS98] bereits ins Jahr 1991 zurückreicht, sind die Arbeiten an dem Projekt noch lange nicht eingestellt. Ziel des *Swedish Institute of Computer Science* (SICS), dem Zentrum der DIVE Entwicklung, ist die Bereitstellung eines Frameworks für das Management großer Datenströme, wie sie etwa von virtuellen Umgebungen hervorgerufen werden. Insofern ist DIVE nicht ausschließlich auf die Übertragung dreidimensionaler Informationen ausgelegt, sondern integriert den Transfer verschiedener Medientypen wie etwa Video, Audio sowie 3D und Internet Informationen. Anwendungsgebiete liegen neben dem Austausch von Informationen in den Bereichen Training und Simulation. Eine *Cave Automatic Virtual Environment* (CAVE) [CNSD93] Anbindung existiert ebenfalls.

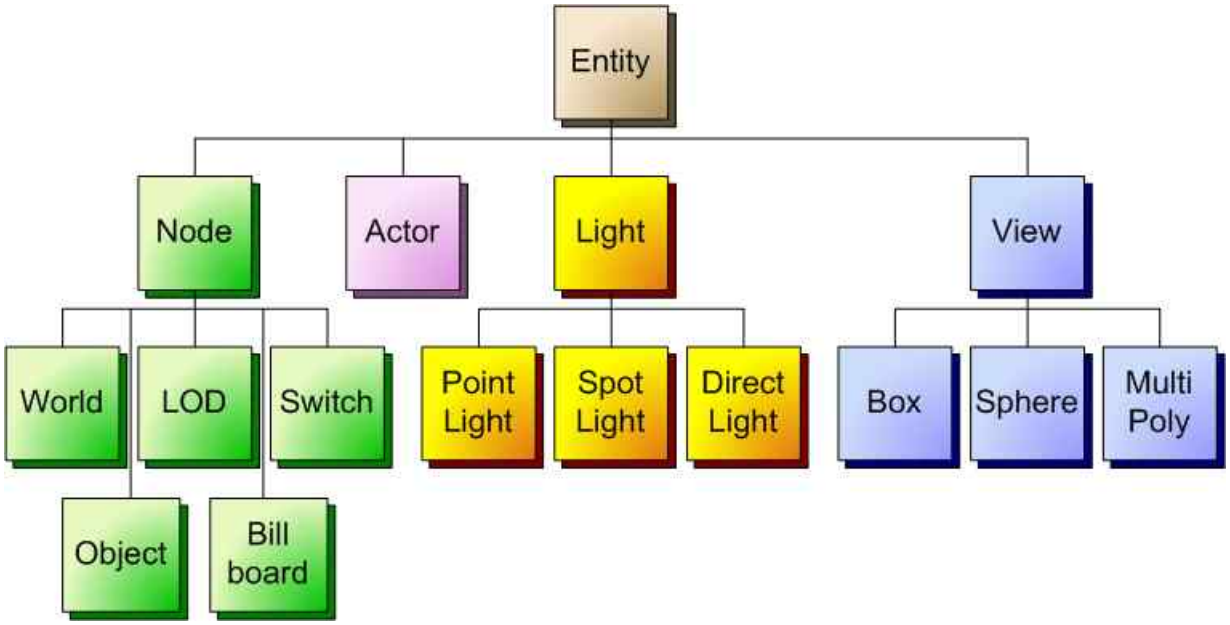

Abbildung 5.2: *Das grundlegende Element einer DIVE Welt ist eine Entity, welche neben visuellen Informationen auch benutzerdefinierte Daten und Verhaltensbschreibungen beinhalten kann.*

Kernstück der DIVE Architektur ist die verteilte Welt-Datenbank. Sämtliche Applikationen kommunizieren nicht direkt untereinander, sondern verwenden die Abstraktion der durch die Welt-Datenbank angebotenen Schnittstelle als gemeinsames Medium. Eine DIVE Welt besteht aus einer hierarchischen Anordnung sogenannter *Entities* (siehe Abbildung 5.2). Eine Entity kann sowohl visuelle Informationen als auch benutzerdefinierte Daten und Verhaltensbeschreibungen umfassen. Letztere werden mit Hilfe der *Tool Command Language* (TCL) [HM97] formuliert und über vom System ausgelöste Ereignisse getriggert. Ereignisquellen können dabei beispielsweise Benutzerinteraktionen, Timeouts oder Kollisionen sein. Jeder Teilnehmer einer Szene wird als *Actor* bezeichnet. Sie repräsentieren entweder einen menschlichen Benutzer oder den Prozess einer Applikation, welche auf die Daten einer Welt zugreift. Weitere Begriffe für die Darstellungen menschlicher Benutzer sind *Body Icon* beziehungsweise Avatar (siehe Abschnitt 3.2). Eine Welt bleibt solange erhalten bis der letzte Actor die Welt verlassen hat. In diesem Fall wird der Zustand einer Welt eingefroren und

bei erneutem Bedarf wieder initialisiert. Die Repräsentation der 3D Szenen selbst erfolgt mit Hilfe des in Abschnitt 4.7 erwähnten IRIS Performer Szenegraphen.

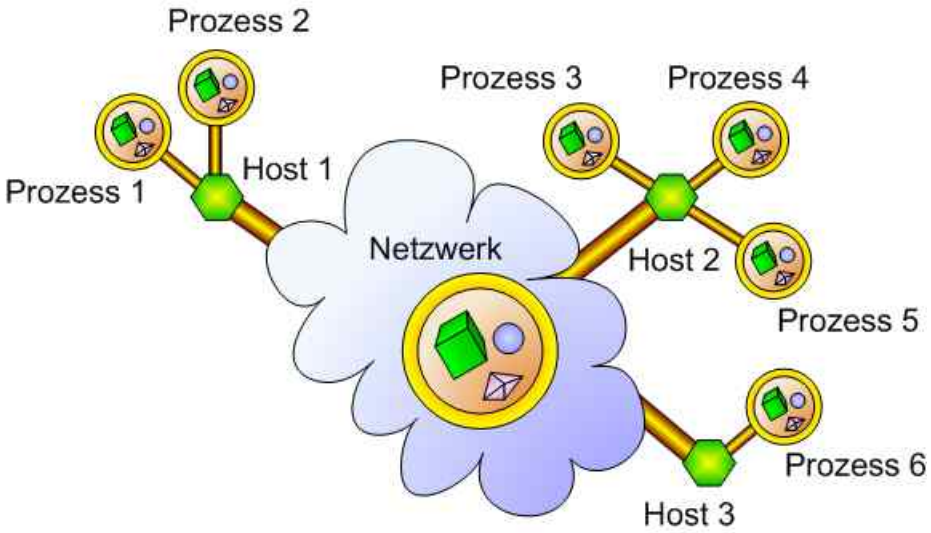

Abbildung 5.3: *DIVE basiert auf einer Peer-to-Peer Topologie, wobei jeder Host zumindest eine teilweise Replikation der Welt-Datenbank erhält.*

Wie in Abbildung 5.3 illustriert, beruht DIVE im Gegensatz zu vielen anderen Ansätzen im Bereich der virtuellen Umgebungen auf einer Peer-to-Peer Architektur (siehe Abschnitt 4.8). Ein *Peer* repräsentiert den Prozess einer Applikation, welcher auf einem bestimmten *Host* residiert und mit dem DIVE Netzwerk verbunden ist. DIVE verwendet das Prinzip der aktiven Replikation, d.h. jeder Prozess hält eine Kopie der für ihn relevanten Daten. Dieses Konzept ist nicht mit der in Abschnitt 5.1.5 beschriebenem Selektion der Daten anhand einer Area-of-Interest zu verwechseln. Vielmehr kann der Benutzer aus der hierarchischen Datenbank einen Teilbaum auswählen, der dann komplett auf den betreffenden Peer kopiert wird. Da möglicherweise mehrere Benutzer ein Interesse am gleichen Teil der Hierarchie haben, besteht die Option, eine Gruppe zu bilden und mit dem Teilbaum über eine *Multicast* Verbindung zu assoziieren. Eine derartige Gruppe wird auch als Leichtgewichtsgruppe bezeichnet. Nimmt ein Benutzer Änderungen an einer Welt vor, so wirken sich diese erst einmal auf seine lokale Kopie aus. Da sich die Kopie im Speicher des jeweiligen Rechners befindet, erhält der Benutzer ein sehr schnelles Feedback auf seine Manipulationen. Danach werden die Änderungen den übrigen Teilnehmern per Multicast mitgeteilt. Mit dieser Vorgehensweise akzeptiert DIVE kurzfristige Inkonsistenzen zwischen den einzelnen Duplikaten. Beim Versenden der Multicasts unterscheidet DIVE zwischen sicheren und kontinuierlichen Datenströmen. Der Schwerpunkt der sicheren Datenströme liegt nicht in der Verschlüsselung der Informationen, sondern vielmehr in einer *Quality of Service*. Ziel ist die konsistente Repräsentation identischer, replizierter Welten. Geht etwa eine Nachricht zur Transformation eines Elements bei einem Multicast verloren, so versucht der betroffene Peer, diese Informationen mit Hilfe des nächstgelegenen Peers zu reproduzieren. Kontinuierliche Datenströme sind nicht auf die Aufrechterhaltung der Konsistenz replizierter Welten ausgelegt. Ihre Aufgabe ist die verständliche Präsentation von Audio- und Videosequenzen. Hier würde eine ständige Synchronisation zwischen den einzelnen Prozessen des Netzwerks zu einer massiven Qualitätsminderung, wenn nicht gar zur Unverständlichkeit der Daten führen.

Abbildung 5.4: *DIVE Anwendungen dienen unter anderem zu Trainings- und Simulationszwecken.*

Das entscheidende Merkmal, durch das sich DIVE letztlich auszeichnet, ist die zugrunde liegende Peer-to-Peer Architektur. Diese bietet den Vorteil, die Rechenlast nicht schwerpunktmäßig einem einzelnen Endgerät wie etwa einem Server aufzubürden, sondern sie auf alle Teilnehmern in gleicher Weise zu verteilen. Ein Problem von Peer-to-Peer Ansätzen liegt in der gegenüber Client-Server-Konzepten schwierigeren Synchronisation dynamischer Vorgänge, da keine zentrale, permanent konsistente Repräsentation der Szene existiert. DIVE geht in diesem Fall das Risiko kurzfristiger Inkonsistenzen ein, was für einige Szenarien wie etwa Virtual Chatrooms vollkommen ausreichend sein mag. Dies gilt jedoch nicht für Anwendungen, bei denen es auf Geschicklichkeit und schnelle Reaktionen infolge unerwarteter Ereignisse ankommt. Beispielsweise ist der agierende Benutzer aufgrund der kurzen Antwortzeiten seiner lokalen Kopie in 3D Spielen oder interaktiven Trainingssimulationen stets in Vorteil. Größtes Manko von DIVE ist aber die zumindest teilweise Replikation der Datenbanken mit all ihren Inhalten wie Geometrie, Farben oder Texturen. DIVE geht hier nicht nur von einer Gleichberechtigung der einzelnen Peers, sondern auch von einer ungefähr identischen Rechenleistung der betreffenden Endgeräte aus. Da also keine Anpassung der Daten existiert, sind schwächere Endgeräte mit den anfallenden Datenmengen oftmals überfordert. Die progressive Übertragung der Daten, wie in Abschnitt 5.1.1 verlangt, ist auf das Streamen von Audio und Video beschränkt, gleichwohl Performer immerhin die Möglichkeit mehrerer Detailstufen vorsieht. Um nun den Einsatz von DIVE auf einem breiten Spektrum an Endgeräten zu erlauben, geben die schwächeren Endgeräte die maximale Datenmenge vor. Als Konsequenz aus der mangelnden Adaptivität verfügen DIVE Welten in der Regel

über keine hohe Grafikqualität. Die Verwendung von IRIS Performer für die Darstellung der 3D Szenen ist aufgrund der hohen Optimierungsmöglichkeiten auf die vorhandene Hardware durchaus ein geeignetes Mittel. Wie fast alle Szenegraphen verlangt IRIS Perfomer jedoch die Bereitstellung verschiedener Komponenten zur Verwaltung dynamischer Szenen, d.h. Lösungen wie sie in Abschnitt 5.1.1 spezifiziert werden. Standardmäßig bietet Performer keine ausgabesensitive Visualisierung, sondern lediglich ein View Frustum Culling mit Hilfe von Bounding Volumes (siehe Abschnitt 4.6). Hier besteht somit der Bedarf an der in Abschnitt 5.1.4 beschriebenen Technik.

5.2.2 RING

RING [Fun95] ist ein Client-Server-System für Virtual Environments mit hoher Verdeckungstiefe (siehe Abschnitt 4.6), welches im Jahr 1995 von Thomas Funkhouser vorgestellt wurde. Primäre Anwendungsszenarien sind also Städtemodelle oder die von Funkhouser als Beispiel herangezogenen Innenarchitekturen. Die Hauptintention des Ansatzes liegt nicht in der adaptiven Übertragung visueller Informationen, sondern vielmehr in der Reduktion des Nachrichtenaustauschs aufgrund der konsistenten Repräsentation verteilter, dynamischer Szenen. Was Funkhousers Verfahren für die vorliegende Arbeit so interessant macht, ist der erstmalige Einsatz visueller Kriterien für die Selektion wichtiger zu übertragender Informationen. Die grundlegende Idee ist dabei, Änderungen innerhalb einer virtuellen Welt nur an diejenigen Teilnehmer weiterzuleiten, welche die Änderungen auch optisch wahrnehmen können. Funkhouser verspricht mit dieser Vorgehensweise einen bis zu 40mal geringeren Aufwand im Vergleich zu konventionellen Verfahren. Hierunter sind beispielsweise Ansätze mit Punkt-zu-Punkt oder Broadcast Topologien mit N Knoten zu verstehen, welche in jedem Zeitschritt einer Simulation $O(N^2)$ beziehungsweise $O(N)$ Nachrichten für die Aktualisierung der N verteilten Repräsentationen einer Szene versenden (siehe Abschnitt 4.8).

Der Ansatz von Funkhouser basiert sehr stark auf dem von Teller et al. vorgestelltem PVS Verfahren [TS91]. Hierbei wird die 3D Szene in Zellen unterteilt und für jede Zelle eine konservative Sichtbarkeitsmenge bestimmt, welche die von der aktuellen Zelle aus sichtbaren Nachbarzellen identifiziert. Die Größe der Sichtbarkeitsmengen ist in Funkhousers Beispiel durch Portale beschränkt. Äquivalent zu DIVE repräsentiert RING eine Welt als eine Menge sogenannter Entities, welche über visuelle und verhaltensspezifische Informationen verfügen (siehe auch Ansatz der Animationselemente in Abschnitt 4.2). RING unterscheidet zwischen statischen und dynamischen Entities. Erstere beschreiben die Umgebung der virtuellen Welt wie etwa die Landschaft oder die Gebäude. Dynamische Entities besitzen ein Verhalten und werden ähnlich dem Konzept der DIVE Actors entweder durch einen Benutzer oder durch eine Applikation gesteuert. Entsprechend kann es sich bei den dynamischen Entities etwa um Avatare oder Fahrzeuge handeln. Im Gegensatz zu Virtual Chatrooms legt Funkhouser aber keinen großen Wert auf die visuelle Repräsentation dieser Avatare, sondern verwendet für die Entities anderer Clients auf einem Endgerät eine vereinfachte Darstellung. Überhaupt vermeidet Funkhouser die Übertragung visueller Informationen und geht davon aus, dass die entsprechenden Daten bereits auf den Clients vorhanden sind. In seinem Ansatz benötigen die Server keine visuellen Daten wie etwa Punkte, Polygone oder Texturen. Stattdessen reicht eine Speicherung der Zellen mit einigen Zusatzinformationen bezüglich der konservativen

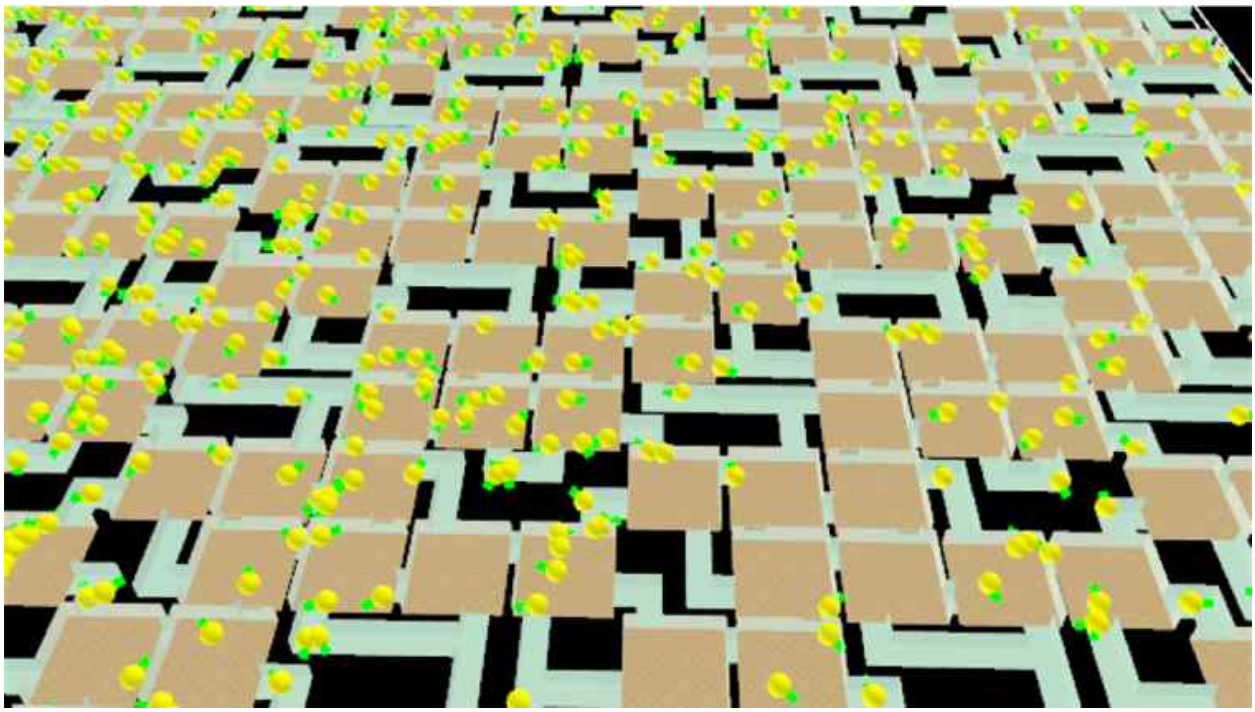

Abbildung 5.5: *RING erreicht eine Reduktion des Nachrichtenaufwands durch das Einsetzen visueller Kriterien. Die Beispielszene entspricht einer Innenraumarchitektur mit extremer Verdeckungstiefe. Insofern ist das hier verwendete PVS Verfahren sehr gut geeignet (Screenshot des RING Konzepts [Fun95]).*

Sichtbarkeitsmenge sowie die vereinfachte Repräsentation aller dynamischen Entities der Szene.

RING erlaubt keine direkte Kommunikation zwischen den Clients. Vielmehr senden die Clients eventuelle Änderungen ihrer dynamischen Entities an einen Server, welcher die Informationen weiterleitet. Dabei muss es sich nicht zwangsläufig um Clients handeln, sondern es können auch andere Server das Ziel der Informationen sein. Entsprechende Situationen treten beispielsweise in einer Server-Hierarchie auf. Der Umweg über den Server hat eine Reihe von Vorteilen:

- Der Client muss lediglich eine Nachricht an den Server versenden und nicht $N - 1$ Nachrichten an die übrigen Clients. Dies ist vor allem eine Entlastung für schwächere Endgeräte.
- Der Server kann die Informationen filtern und nur an diejenigen Clients weiterleiten, die auch ein Interesse an der Information haben. Eine entsprechende Filterung kann etwa mit Hilfe visueller Kriterien geschehen.
- Der Server ist nicht nur in der Lage, die Nachrichten zu filtern, sondern er kann sie zusätzlich um andere wichtige Informationen ergänzen beziehungsweise mit Hilfe seiner Kenntnisse eventuell vereinfachen.

Neben diesen aufgelisteten Punkten bestehen im Falle hierarchischer Server-Systeme die in Abschnitt 4.8 beschriebenen Vor- und Nachteile: Einerseits kann die Belastung auf mehrere

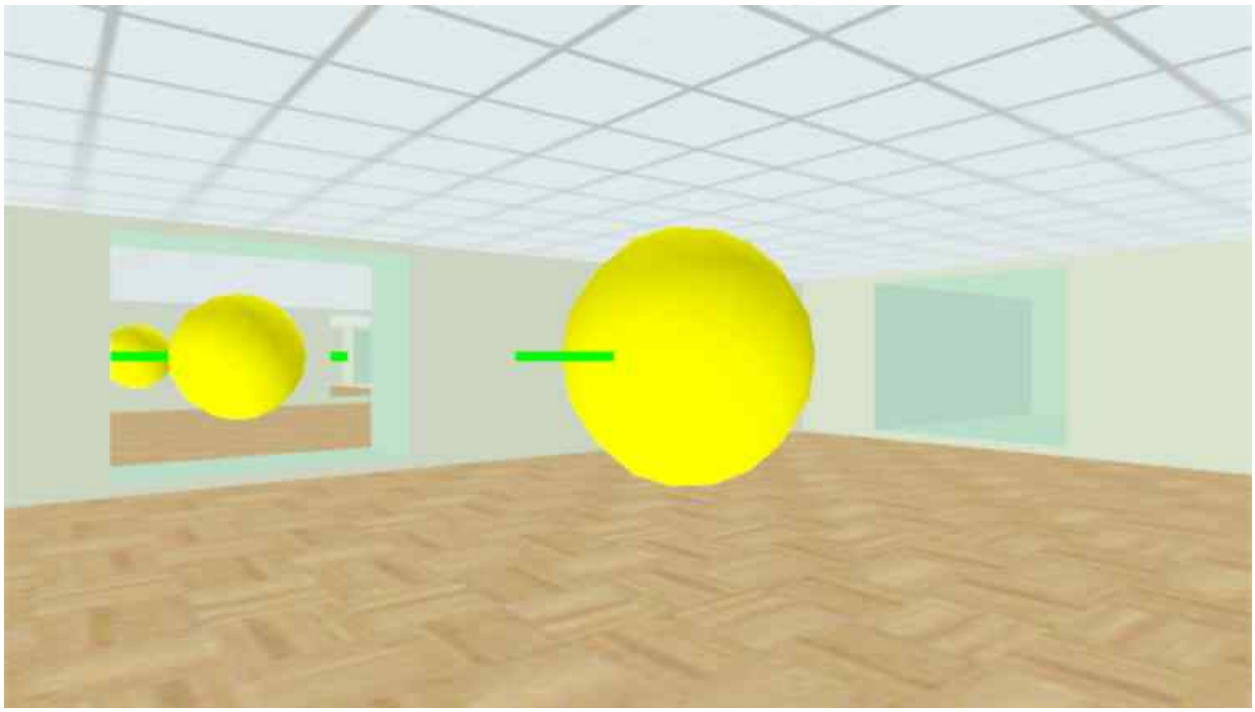

Abbildung 5.6: *Sobald ein Avatar in den Sichtbereich eines anderen Avatars gerät, müssen die mit den Avataren assoziierten Clients jeweils über die Bewegungen des anderen informiert werden (Screenshot des RING Konzepts [Fun95]).*

Server verteilt werden, andererseits erhöhen sich aber auch die Latenzzeiten der Nachrichten im Vergleich zu einer direkten Client-Client Kommunikation.

Nach dem RING Konzept bieten die Server die Möglichkeit, einem Client nur diejenigen Informationen zukommen zu lassen, die sich auf den Sichtbereich des betreffenden Clients auswirken. Jedesmal, wenn die Entity eines Clients C_A eine Zelle betritt, welche zur konservativen Sichtbarkeitsmenge eines Clients C_B gehört, so wird C_B mittels einer *Add* Nachricht unterrichtet. Verlässt der Client C_A den potentiellen Sichtbereich von C_B wieder, so erfolgt eine *Remove* Nachricht. Zwischen diesen beiden Nachrichten hält der Server C_B mit ständigen Aktualisierungen bezüglich C_A auf dem Laufenden. Befinden sich in der konservativen Sichtbarkeitsmenge eines Clients C_i mit $i = 1, \ldots, N$ momentan k_i Clients, dann muss der Server anstelle von N lediglich k_i Nachrichten versenden. Dabei ist aufgrund der hohen Verdeckungstiefe der Szene zu erwarten, dass k_i deutlich kleiner ausfällt als N.

Der ausschlaggebende Punkt an Funkhousers Arbeit ist der Beweis, dass mit Hilfe visueller Kriterien der Nachrichtenaufwand nicht nur für die Übertragung visueller Informationen, sondern auch für die Aktualisierung verteilter, dynamischer Szenen massiv reduziert werden kann. Sein Schwerpunkt ist nicht die Adaption der visuellen Informationen an die Leistungsmerkmale eines Endgerätes. Da Funkhouser den Beweis mit Hilfe eines PVS Verfahrens führt, unterliegt er als Konsequenz den Nachteilen dieser Technik: Eine Veränderung der Wände oder Gebäude einer Szene erfordert eine teure Neuberechnung der betroffenen Sichtbarkeitsmengen. Insofern sind denkbare Szenarien auf die in Abschnitt 4.6 erwähnten Anwendungen beschränkt. Der Vorteil von Funkhousers Beispiel liegt in der hohen Verdeckungstiefe der Sze-

ne, die seine Ergebnisse und damit seine Intention verdeutlicht. In anderen Szenarien wird die Differenz $N - k_i$ sicherlich geringer ausfallen. Hier gilt zumindest für den Server ein ähnliches Kriterium wie im Falle der Occlusion Culling Verfahren: Der Aufwand zur Bestimmung der k_i Clients in der konservativen Sichtbarkeitsmenge eines Clients C_i einschließlich dem Versenden der k_i Nachrichten muss weniger Laufzeit beanspruchen als einfach N Nachrichten an alle Clients zu verschicken. Es darf aber nicht übersehen werden, dass die versendeten Nachrichten auch Auswirkungen auf die Belastung der jeweiligen Clients haben.

5.2.3 Virtual Reality Modeling Language

Wie der Name *Virtual Reality Modeling Language* (VRML) [Con] bereits besagt, ist VRML eine Sprache zur Modellierung virtueller Welten. Der Gedanke hinter VRML beziehungsweise hinter virtuellen Welten überhaupt ist der Austausch von Daten in einer dem Menschen vertrauten Art und Weise. Anstelle eines Maschinen-basierten Konzepts steht der Mensch im Mittelpunkt, der durch die Szene navigieren, andere Charaktere wahrnehmen und mit diesen kommunizieren kann. Der Ursprung von VRML reicht bis in das Jahr 1994 zurück, genauer bis zum Zeitpunkt der ersten *World Wide Web* Konferenz in Genf. Infolge der Diskussion über eine *Virtual Reality* Schnittstelle reifte dort der Entschluss, eine gemeinsame Sprache zur Spezifikation virtueller Welten zu entwickeln. Die anfängliche Bezeichnung *Virtual Reality Markup Language* ging später in den auch heute noch gängigen Begriff der Virtual Reality Modeling Language über. Anstelle eines kompletten Neuentwurfs begann die Suche nach schon existierenden Technologien, deren Ergebnis im Verwenden des ASCII-basierten Open Inventor Formats [SGIb] bestand. Aus diesem Grund wurde VRML als eine Untermenge von Open Inventor konzipiert und um Netzwerk Operationen wie etwa *Hyperlinks* zu Dokumenten, FTP Verzeichnissen oder anderen VRML Welten ergänzt. Hier weist VRML einige Querverweise zur *HyperText Markup Language* (HTML) auf, was nicht weiter erstaunlich ist: Die ursprüngliche Idee war nämlich, einen normalen Internet Browser und

Abbildung 5.7: *Cybertown ist eine mit VRML erstellte virtuelle Umgebung (Screenshot von Cybertown [Net]).*

VRML Konsortium		
Construct Internet Design	Apple Computer	dFORM
Black Sun Interactive	Axial Systems	3Dlabs
First Virtual Holdings	Division	IBM
Integrated Data Systems	Superscape	Intel
Mitsubishi Electric Research Labs	Kinetix	S3
Netscape Communications	ParaGraph	Oracle
Intervista Software	Silicon Graphics	Sense8
Template Graphics Software	Microsoft	Sony
Visible Decisions		

Tabelle 5.1: *Die Hauptträger des VRML Konsortiums im Jahre 1996.*

einen VRML Browser parallel zu benutzen, ersteren für die Anzeige von HTML Dokumenten und letzteren für die Begehung virtueller Welten. Beide Browser sollten hierzu miteinander kommunizieren, um die jeweils erforderlichen Dokumente anzeigen zu können. Mittlerweile existieren jedoch auch hybride Browser, die beispielsweise dazu in der Lage sind, virtuelle Welten innerhalb von HTML Dokumenten zu visualisieren. Im Jahre 1996 wurde die VRML 2.0 Spezifikation zu einem de facto Standard für die Übertragung von 3D Informationen, was aufgrund des illustren Kreises an bedeutenden Firmen innerhalb des VRML Konsortiums ebenfalls nicht überraschend ist (siehe Tabelle 5.1). Bei der Veröffentlichung der Spezifikation während der SIGGRAPH '96 Konferenz stimmte das *International Standards Organization* (ISO) Komitee einer Anerkennung als ISO-Standard zu. Der endgültige Entwurf erschien dann 1997 als VRML97 Spezifikation. Später entstand aus dem ehemaligen VRML Konsortium das jetzige Web3D Konsortium, welches sich für die Spezifikation von *Extensible 3D* (X3D) verantwortlich zeichnet.

Während VRML 1.0 noch auf rein statische Szenen ausgelegt war, bietet VRML 2.0 die Option zur Beschreibung von Animationen und Interaktionen. Vordefinierte Animationen können mit Hilfe von *Interpolator* Knoten definiert und durch einen *TimeSensor* Knoten getriggert werden. *Script* Knoten erlauben die Beschreibung von Verhalten und werden zwischen Ereignis-Erzeugern und Ereignis-Empfängern platziert. Ereignis-Erzeuger sind alle Arten von *Sensor* Knoten, beispielsweise der *Collision* Knoten zum Detektieren von Kollisionen und der bereits erwähnte TimeSensor Knoten. Als Ereignis-Empfänger gelten grundsätzlich alle Knoten, die eine Liste eingehender Ereignisse definieren und mit einem Ereignis-Erzeuger verbunden sind. Dieses Konzept deckt User-Element-Interaktionen und Element-Element-Interaktionen ab. Für die visuelle Repräsentation von Modellen bietet VRML neben vordefinierten Körpern wie Würfeln, Kugeln und Zylinder unter anderem die Möglichkeit zur Darstellung von Punktwolken, Polygonnetzen, Höhenfeldern und NURBS. Ein Knotentyp zur Verwaltung mehrerer Detailstufen ist ebenfalls vorhanden. Gesonderte Knoten zur räumlichen Sortierung einer Szene existieren nicht. Hierzu kann jedoch der *Group* Knoten in Kombination mit einer Bounding Box verwendet werden. Die Konzeption eines VRML Szenegraphen entspricht dem in Abschnitt 4.7 beschriebenem Schema. Für die Visualisierung auf den Endgeräten verwendet VRML einen klassischen Client-Server-Ansatz: Die Clients

können sich auf einem Server registrieren und die dort vorhandenen Szenen herunterladen. Für die Visualisierung zeichnet sich der jeweilige VRML Browser auf den Endgeräten verantwortlich. Abbildung 5.8 illustriert den grundlegenden Aufbau eines Browsers. Dieser besteht aus drei Hauptkomponenten, nämlich dem Parser, dem Szenegraphen und der akustischen beziehungsweise visuellen Präsentation. Der Parser übersetzt eine VRML Datei in den entsprechenden Szenegraphen. Dieser besteht nicht nur aus der Beschreibung der Szene an sich, sondern auch noch aus der Komponente, welche die verschiedenen Formen der Interaktion auswertet und verarbeitet. Die Präsentations-Komponente traversiert den Szenegraph und kommuniziert das Resultat dem Benutzer.

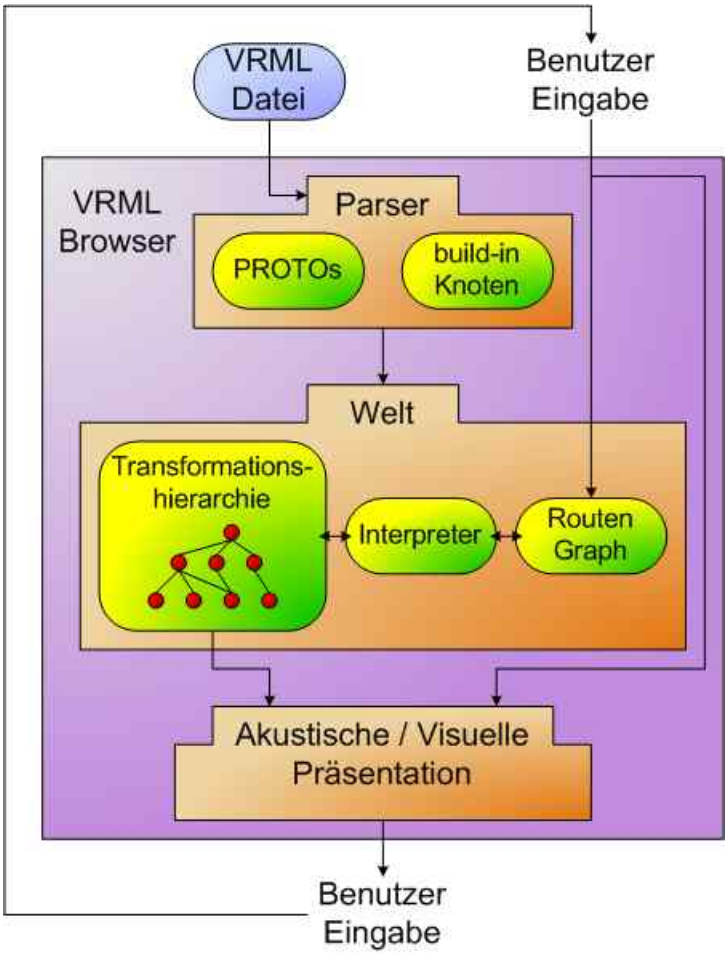

Abbildung 5.8: *Ein VRML Browser besteht im Wesentlichen aus drei Teilen. Der Parser liest eine Datei ein und erzeugt daraus einen Szenegraphen. Dieser repräsentiert die Szene und berechnet zudem mögliche Interaktionen und Animationen. Die Präsentation rendert den Szenegraphen für die akustische und visuelle Ausgabe.*

Auch wenn immer noch eine ganze Reihe von Anwendungen auf VRML basieren (z.B. Cybertown [Net]), ist es nach der anfänglichen Euphorie in den vergangenen Jahren ein wenig ruhiger um VRML geworden. Ausschlaggebend hierfür sind einige Schwachstellen im VRML Konzept, welches mitunter eine Übertragung der gesamten Informationen einer Szene vorsieht. Eine Adaption der Daten findet somit weder in Hinsicht auf die Endgeräte noch auf die vorhandene Bandbreite statt. Letztere hat sich dabei im Besonderen als Flaschenhals für

die anfallenden Datenmengen erwiesen, weshalb trotz steigendem Durchsatz auch derzeitige Applikationen unter einer eher schlichten Qualität der Grafik zu leiden haben. Abgesehen von diesem grundlegenden Problem ist die Offenheit das Systems sowohl Stärke als auch Schwäche von VRML: Einerseits können neue Techniken gleich einem Baukasten-Prinzip integriert werden, andererseits bietet VRML abgesehen von den Schnittstellen aber auch keine Lösungen an. Ein Beispiel hierfür ist das Problem der Schnittelemente innerhalb von Raumunterteilungsbäumen und die konsistente Pflege dieser Bäume im Falls dynamischer Szenen. Ausgabesensitive Visualisierungen obliegen der Implementierung des jeweiligen Browsers. Dies ist ein schwieriges Unterfangen, da weder eine genaue Spezifizierung für Occluder noch für eine räumliche Sortierung der Szene vorhanden ist. Letztlich kann der Browser nur auf ein geeignetes Arrangement der Szene mit Hilfe der Group Knoten hoffen. Die Aufgabe der VRML Webseite und der Übergang zum Web3D Konsortium ist wohl ein Indiz, dass mit X3D ein Neuanfang gewagt werden soll. Für die Übertragung von 3D Grafik sieht die X3D Spezifikation die Verwendung von Mpeg-4 vor, dessen Beschreibung sich in Abschnitt 5.2.4 findet. Obgleich die X3D Spezifikation nun doch schon mehrere Monate zur Verfügung steht, beschränken sich derzeitige Beispielanwendungen auf das Streamen von interaktiven Videosequenzen. Es bleibt daher abzuwarten, inwieweit es zu einer Verwendung von X3D in 3D Applikationen kommt.

5.2.4 MPEG-4

Die *Moving Picture Expert Group* (MPEG) ist vor allem aufgrund der Entwicklung von Standards wie MPEG-1 und MPEG-2 zur digitalen Kodierung von Filmen bekannt geworden. Die geplante Konzeption von MPEG-3 als Standard für *High Definition TV* floss schließlich 1992 mit in den MPEG-2 Standard ein [9]. MPEG-3 ist zu unterscheiden von MP3, dem *MPEG Audio Layer-3* [Hac00], der unter anderem zur Komprimierung der akustischen Informationen in MPEG-1 dient. Wie MPEG-1 und MPEG-2 war auch MPEG-4 eigentlich als Format zur Komprimierung von Filmen, d.h. zur Kodierung von Video- und Audiodaten vorgesehen [PE98]. Aufgrund des wachsenden Bedarfs an der Übertragung multimedialer Daten über das Internet wurde die Zielsetzung von MPEG-4 aber um den Transfer von Audio- und Videosequenzen, Texten sowie zweidimensionaler und dreidimensionaler Grafik erweitert. MPEG-7, auch als *Multimedia Content Description Interface* bezeichnet, ist ein Standard für die Beschreibung von Multimedia-Inhalten zum Zwecke der Suche, der Wiedererkennung, der Organisation, der Auswertung oder der Auswahl nach persönlichen Präferenzen.

Im Rahmen der vorliegenden Arbeit ist lediglich die Intention, MPEG-4 für die Darstellung und Übertragung von interaktiven 3D Szenen zu nutzen, von Interesse. Letztlich ist das hierzu angedachte *Binary Format For Scenes* (BIFS) ein Derivat der in Abschnitt 5.2.3 beschrieben Virtual Reality Modeling Language. Dies mutet ein wenig seltsam an, da ja eigentlich X3D, also der Nachfolger von VRML 2.0, MPEG-4 für die Übertragung von 3D Szenen verwendet. Die Erweiterungen gegenüber VRML beschränken sich auf die Definition von 2D Objekten und Gesichtsanimationen sowie auf die Verbesserung der Audio-Schnittstelle.

[9] High Definition TV ist auch unter der Bezeichnung High-resolution Digital Television (HDT) bekannt. Ziel ist unter anderem, die im Vergleich zu einem Computer Monitor deutlich schlechtere Auflösung der Fernseher auf ein entsprechendes Niveau anzuheben.

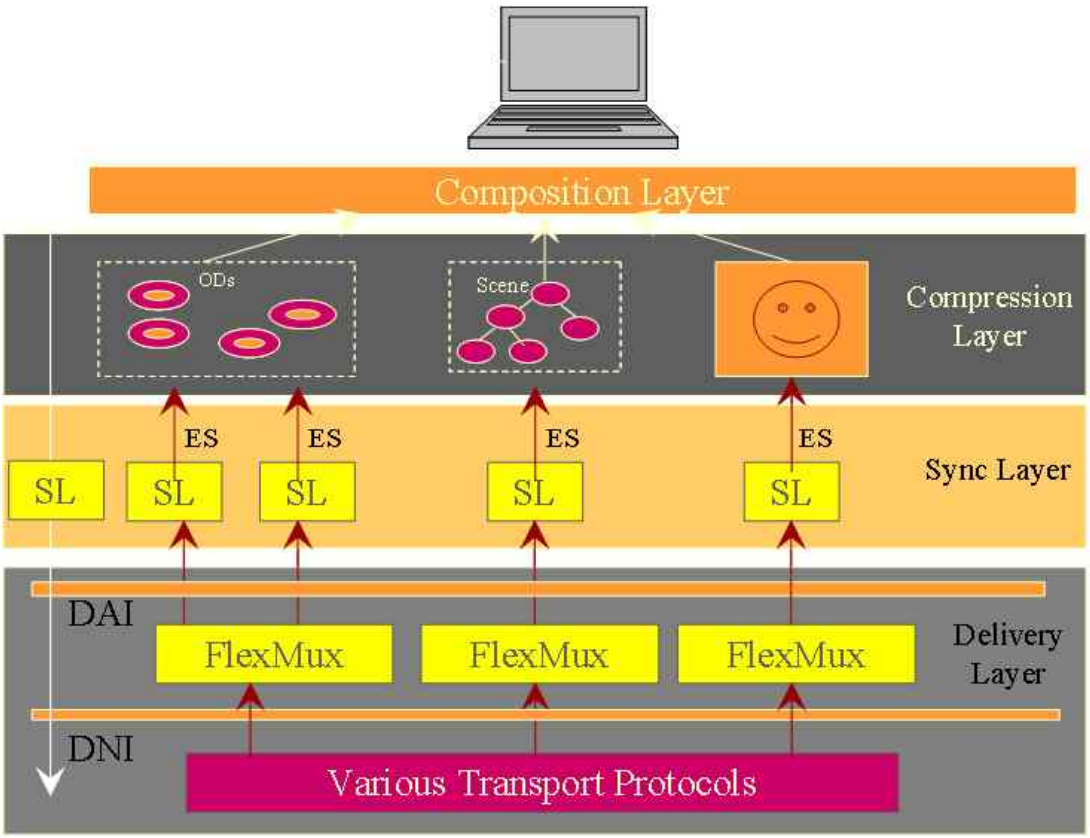

Abbildung 5.9: *Das Schichtenmodell von MPEG-4.*

Ähnlich ist ebenfalls das Konzept zur Verteilung und Manipulation der Szenen auf den einzelnen Endgeräten. Basierend auf einem Client-Server-Ansatz können die Clients die auf einem Server konservierten Szenen herunterladen und begutachten. Elemente dürfen ein- und ausgefügt sowie ganze Szenen durch neue ersetzt werden. Zusätzlich besteht die Möglichkeit, einzelne Werte wie etwa Betrachterpositionen, Matrizen und Lichtparameter zu beeinflussen. Neu an BIFS ist dagegen das dem Namen entsprechende komprimierte binäre Dateiformat, ein Zugeständnis an die Erkenntnisse, welche aus der Übertragung der unhandlichen ASCII-basierten VRML Repräsentationen gezogen wurden. Allerdings geht mit BIFS auch die Option verloren, eine Datei in einem beliebigen Texteditor zu öffnen und zu bearbeiten.

Grundlegender Baustein der MPEG-4 Architektur ist das *Media Object*, welches ein Element der Szene beschreibt. Die Idee der Media Objects ist nicht auf ein bestimmtes Medium fixiert. Vielmehr kann ein Media Object jeden von MPEG-4 unterstützen Medientyp wie etwa Text, Audio oder Video annehmen. Während der Übertragung wird ein Media Object durch einen bis mehrere *Elementary Streams* repräsentiert. Für jedes Media Object existiert ein *Object Descriptor*, welcher die mit dem Media Object assoziierten Elementary Streams verwaltet. Unter anderem beinhaltet der Object Descriptor für jeden Elementary Stream eine Information, welche Codecs dem jeweiligen Datenstrom zuzuordnen sind. Die Gesamtarchitektur von MPEG-4 setzt sich aus vier Schichten zusammen, nämlich aus der *Delivery Layer*, der *Synchronization Layer*, der *Compression Layer* und der *Composition Layer* (siehe Abbildung 5.9). Die Delivery Layer beinhaltet einen *Multiplexer*, der selbst wiederum aus

zwei Schichten, der *Flexible Multiplexer Layer* und der *Transport Multiplexer Layer*, besteht. Letztere hat die Aufgabe, die anfallenden Datenströme zu übertragen. Sie wird in MPEG-4 nicht spezifiziert, da es sich im Wesentlichen um existierende Protokolle zur Übertragung von Daten via Netzwerk sowie um Standards zur digitalen Rundfunk- und Fernsehausstrahlung handelt. Die Flexible Multiplexer Layer ist entsprechend dem *Digital Media Integration Framework* (DMIF) definiert. Sie erlaubt das Gruppieren von Elementary Streams mit ähnlichen Anforderungen, beispielsweise um die Anzahl der Netzwerkverbindungen zu reduzieren. DMIF weist Parallelen zu FTP auf und definiert ein Session Protokoll für das multimediale Streamen mit Hilfe generischer Übertragungstechnologien. Die Synchronization Layer synchronisiert die eingehenden Datenströme mit Hilfe verschiedener Zeitstempel. Innerhalb der Compression Layer werden die Datenströme dem im Object Descriptor eingetragenem Codec zugewiesen. Basierend auf den Informationen der Object Descriptors setzt die Kompositions Layer die Szene aus den verschiedenen Datenströmen wieder zusammen. Hierzu existieren Elementary Streams sowohl zur räumlichen Beschreibung der Szene als auch zur Repräsentation der Media Objects. Da MPEG-4 keine Spezifikation für das Rendern der Szene vorgibt, obliegt dies der jeweiligen Applikation des empfangenden Endgerätes.

Eine Beurteilung von MPEG-4 ist äußerst schwierig. Zwar sind eine Vielzahl verschiedener Dokumente und Spezifikationen verfügbar, leider existieren aber zumindest im Bereich der Übertragung von 3D Szenen keine Beispielanwendungen, die eine Validierung des Konzepts erlauben. Die vorhandenen Beispiele sind eher mit interaktiven Video-Konferenzen vergleichbar. Das Schichtenmodell der Architektur bietet eine hohe Flexibilität in Bezug auf die zugrunde liegenden Plattformen und deren Netzanbindungen. Die einzelnen Schichten ermöglichen eine Auftrennung der Datenströme wie in Abschnitt 5.1.1 verlangt. Hierüber ist die Kodierung der Daten durch spezifische Codecs gesichert. Das alleine ist für eine effiziente Übertragung einer Szene aber nicht ausreichend. Unter anderem muss immer berücksichtigt werden, dass hohe Kompressionsraten in der Regel eine teure Dekompression auf dem empfangenden Endgerät implizieren. Schwächere Endgeräte können dabei leicht überfordert werden. An dieser Stelle ist es oftmals sinnvoller, eine Reduktion der Datenmenge durch eine gute Selektion der Daten zu erreichen. Hier erbt MPEG-4 aber konsequenterweise die Nachteile von VRML, d.h. eine Szene kann nicht gemäß einem Area-of-Interest Konzept übertragen werden (siehe Abschnitt 5.1.5). Eine Ursache dafür liegt in dem gleichfalls fehlenden Konzept zur räumlichen Sortierung einer Szene. Die Reduktion mit Hilfe invasiver Verfahren wird ebenfalls nicht berücksichtigt, allerdings sind für die nächste Version progressive 3D Polygonnetze geplant. Als Fazit bleibt an dieser Stelle festzuhalten, dass MPEG-4 ein flexibles Framework für die Übertragung multimedialer Informationen darstellt. Allerdings steht und fällt das Konzept mit den eingesetzten Komponenten. Der Gebrauch von VRML als eine dieser Komponenten ist sicherlich mit Vorsicht zu betrachten.

5.2.5 System von Teler et al.

Eyal Teler und Dani Lischinski stellten im Jahr 2001 ein System für die Übertragung großer 3D Szenen auf der EUROGRAPHICS vor [TL01]. Im Gegensatz zu den vorangehenden Architekturen ist ihre Intention die selektive Übertragung der visuellen Informationen einer 3D Szene. Hierdurch soll vermieden werden, wie im Falle von DIVE, VRML und MPEG-4 eine

Visualisierung auf einem Endgerät erst nach dem vollständigen Transfer der Szene in Angriff nehmen zu können. Teler et al. gehen aber noch einen Schritt weiter, indem sie die Zeitspanne bis zur Visualisierung nicht nur mit Hilfe einer geeigneten Selektion der Elemente, sondern auch durch die Auswahl verschiedener Repräsentationen eines Elements reduzieren. Welches Element und welche Repräsentation letztlich transferiert wird ist das Ergebnis einer Kosten-Nutzen Analyse: Wie teuer ist die Übertragung eines bestimmten Elements in Relation zum Gewinn an visueller Qualität durch das Vorhandensein des Elements im resultierenden Bild? Teler et al. gehen dabei von der Annahme aus, dass der entscheidende Flaschenhals eines Systems zur Übertragung von 3D Szenen nicht in den Leistungsmerkmalen der beteiligten Endgeräte, sondern in den begrenzten Bandbreiten der Netzwerke seinen Ursprung hat. Demnach kann ein Endgerät alle ihm zugesendeten Informationen verarbeiten.

Der Ansatz von Teler et al. basiert auf einem Client-Server-System, wie es schon in VRML und MPEG-4 zum Einsatz kommt: Ein Benutzer kann sich mit seinem Endgerät bei einem Server anmelden und eine dort vorhandene 3D Szene zur interaktiven Begehung auswählen. Die Selektion der zu übertragenden Elemente erfolgt mit Hilfe eines View Frustum Cullings (siehe Abschnitt 4.6), d.h. lediglich Elemente innerhalb der Sichtpyramide des Betrachters werden berücksichtigt. Die Elemente einer Szene sind als Polygonnetze modelliert und liegen auf dem Server entweder in einem einzigen Mesh, in einigen vorberechneten Detailstufen oder in einem progressiven Format vor. Teler et al. verweisen für letzteres auf den Ansatz von Hoppe [Hop96], wenngleich eine entsprechende Implementierung in ihrem System noch nicht vorhanden ist. Sobald ein Element übertragen werden soll, kommt zu den drei genannten Repräsentationen eines Elements noch eine weitere mögliche Darstellung hinzu: Da der Server den Betrachterstandort und die Blickrichtung eines Clients kennt, rendert er anhand dieser Informationen lokal ein Bild des Elements, welches dann an den Client übertragen und dort eingeblendet wird. Derartige Techniken sind auch unter dem Begriff der *Imposter* [SLS+96] bekannt und sind vergleichbar mit der Aufnahme eines Fotos von dem betreffenden Element [10]. Die Intention hinter der Verwendung der Imposter ist ein im Vergleich zu rein polygonalen Ansätzen besseres Verhältnis zwischen Reaktionszeit und Bildqualität. Da die resultierenden Imposter bei geringeren Entfernungen zum Betrachter aufgrund ihrer begrenzten Auflösung deutlich an Qualität verlieren beziehungsweise die Generierung von Imposter mit entsprechender Auflösung immer mehr Rechenleistung beansprucht, werden sie im Laufe der Navigation durch eine schrittweise Übertragung der progressiven Daten oder Detailstufen ersetzt. Somit kann für ein Element sowohl eine Polygon- als auch eine Imposter-basierte Darstellung auf einem Client existieren. Diesem obliegt dann während der Visualisierung die Aufgabe, für jedes Element die optisch beste Lösung zu bestimmen.

Die Berechnung der Kosten-Nutzen Analyse erfolgt ähnlich zu Funkhouser et al. [FS93] und Maciel et al. [MS95] anhand der folgenden vier Kriterien:

- **Genauigkeit**: Die Genauigkeit ist ein Maß, wie gut die gewählte Repräsentation ein Element im Vergleich zu dessen vollständig verfügbaren Daten während der Visualisierung approximiert. Die Genauigkeit ist kein statisches Merkmal, da beispielsweise

[10] Durch die Verwendung der Imposter weist der Ansatz von Teler et al. Parallelen zu SGI VizServer [SGId] auf, wobei die SGI Applikation allerdings nicht nur einzelne Elemente, sondern ganze Frames als Bilder generiert und diese per Video an die Clients versendet.

ein in Bezug auf einen Betrachterstandort erzeugter Imposter nur von der betreffenden Position aus eine Genauigkeit von 100 Prozent aufweist. Geht der Betrachter auf den Imposter zu, so verliert dieser an Genauigkeit. Wegen der Reduktion der dreidimensionalen Elemente zu zweidimensionalen Imposter, gilt gleiches auch bei einer Änderung des Blickwinkels.

- **Sichtbarkeit**: Für die Sichtbarkeit eines Elements sind mehrere Faktoren verantwortlich. Zum einen muss das Element zumindest teilweise innerhalb der Sichtpyramide des Betrachters liegen und zum anderen darf es nicht von anderen Elementen vollständig verdeckt werden. Weitere Kriterien sind Nebel und Bewegungsunschärfen (siehe Abschnitt 4.3).

- **Bedeutung**: Die Bedeutung eines Elements hängt davon ab, welche Aufmerksamkeit ein Betrachter dem Element schenkt beziehungsweise schenken sollte. Beispielsweise ist ein frontal auftauchendes Hindernis oftmals wichtiger als ein Berg im Hintergrund, obgleich letzterer wahrscheinlich einen größeren Bildbereich beansprucht.

- **Sichtbarkeit von Änderungen**: Dieses Kriterium bezieht sich auf die Sichtbarkeit der Wechsel zwischen den einzelnen Repräsentationsformen oder Detailstufen. In einer optimalen Lösung sind derartige Vorgänge für den Betrachter vollständig transparent.

Da die Kriterien teilweise nur sehr schwierig zu bestimmen sind, nehmen Teler et al. einige Einschränkungen vor. So berücksichtigen sie für das Kriterium der Sichtbarkeit lediglich die Größe eines Elements und dessen Position in Bezug zur Sichtpyramide (View Frustum Culling). Das Kriterium der Bedeutung findet aufgrund seiner Abhängigkeit von der zugrunde liegenden Applikation gar keine Beachtung. Für die Genauigkeit der Imposter- und Polygon-basierten Repräsentationen geben Teler et al. Abschätzungen an. Hinsichtlich der Imposter rührt folgende Formel aus experimentellen Erkenntnissen her, wonach selbst bei Änderungen des Blickwinkels um 30° den Testpersonen subjektiv keine Qualitätsminderung eines Imposters aufgefallen war:

$$\min\left(1, \frac{h_0}{h}\right) \cdot \left(\frac{1}{\sin 60} \min(\sin\alpha, \sin 60)\right)^2 \tag{5.1}$$

In dieser Abschätzung repräsentiert h_0 die Höhe des Imposters in Pixel, h die Höhe der Projektion des Elements auf die Projektionsfläche und α den Winkel zwischen der Grundlinie des Imposter und der Linie vom Augpunkt des Betrachters zum Zentrum des Imposter. Je größer h im Vergleich zu h_0, desto geringer ist aufgrund der groben Rasterizierung die Genauigkeit des Imposter. Bei seitlichen Blickwinkeln sinkt die Genauigkeit des Imposter ebenfalls, wohingegen Blickwinkel von 60° bis 120° eine hundertprozentige Genauigkeit bedeuten. Die nächste Formel schätzt die Genauigkeit bezüglich einer Polygon-basierten Darstellung:

$$\sqrt{\frac{\max\left(\frac{a}{F}, 1\right)}{\max\left(\frac{a}{f}, 1\right)}} \tag{5.2}$$

Hier beschreibt a die Fläche der projizierten Bounding Box des Elements in Pixel, F die Anzahl der Polygone im Orignalmesh $\hat{M}$ und f die Anzahl der Polygone in der momentanen Detailstufe, weshalb $f \leq F$ und somit $\frac{a}{F} \leq \frac{a}{f}$. Über die Bildung der Maxima mit 1 werden durchschnittliche Polygongrößen von weniger als einem Pixel ignoriert, da sie aufgrund der diskreten Auflösung der Bildschirme keinen visuellen Einfluss mehr haben. Verringert sich der Bildbereich eines Elements derart, dass $\frac{a}{F} \leq 1$, so steigt die Genauigkeit des Modells mit $1/\frac{a}{f}$. Sobald auch $\frac{a}{f} \leq 1$, dann hat die Detailstufe die volle Genauigkeit erreicht. Weil die Imposter-basierte Abschätzung auf der eindimensionalen Höhe beruht, die polygonbasierte Abschätzung aber auf der zweidimensionalen Fläche, wird aus letzterer zusätzlich die Wurzel gezogen.

Das Ziel von Teler et al. ist, mit Hilfe dieser Abschätzungen für die Übertragung eines Elements eine geeignete Reihenfolge sowohl in Hinsicht auf die Art der Repräsentation als auch gegebenenfalls in Hinsicht auf die Detailstufe zu bestimmen, so dass im Laufe der Navigation eine maximale Genauigkeit erreicht wird. Sei nun r_{ij} die j-te Repräsentation eines Elements i zusammen mit einer entsprechenden Übertragungsdauer d_{ij} und einer durch r_{ij} bedingten optischen Qualitätssteigerung b_{ij} bezüglich der Visualisierung auf dem Endgerät. Da das Maß der Qualitätssteigerung sich aufgrund der Navigation des Betrachters mit der Zeit verändert, ist $b_{ij}(t)$ als Funktion über die Zeit t zu beschreiben. Unter der Annahme, dass der Client zu einem Zeitpunkt t bereits mehrere Repräsentationen für ein Element i empfangen hat, kann dort die Qualitätssteigerung nicht nur in Bezug auf eine spezifische Repräsentation, sondern auf das Element selbst definiert werden.

$$b_i(t) = \max_{j \in R_i(t)} (b_{ij}(t)) \tag{5.3}$$

Dabei ist $R_i(t)$ die Menge aller Repräsentationen eines Elements i, die zum Zeitpunkt t auf einem Client verfügbar sind. Aufgrund der begrenzten Bandbreite muss die Summe der Übertragungszeiten aller auf einem Client vorhandenen Repräsentationen kleiner als die aktuelle Laufzeit t sein.

$$\sum_i \sum_{j \in R_i(t)} d_{ij} \leq t \tag{5.4}$$

Wenn der Pfad des Betrachters während seiner Navigation durch die Szene bereits im voraus bekannt ist, dann wird eine Reihenfolge der zu übertragenden r_{ij} gesucht, welche die Qualitätssteigerung aller Elemente für die Dauer der Navigation bis zu ihrem Ende t_{ende} maximiert.

$$\sum_i \int_{t=0}^{t_{ende}} b_i(t) dt \tag{5.5}$$

Da aber außer bei Kamerafahrten weder der Pfad des Betrachters noch die Zeitdauer der Navigation bekannt sind, muss der Server die zu erwartende Qualitätssteigerung durch eine Repräsentation r_{ij} gegen die bereits auf dem Client vorhandene Qualität abschätzen, welche

Formel 5.3 beschreibt. Für diese Abschätzung ergibt sich unter Berücksichtigung, dass auch keine Qualitätssteigerung möglich ist, $\max(b_{ij}(t)-b_i(t),0)$ und somit aus Formel 5.5 folgendes Integral mit unendlicher Zeitdauer:

$$\int_{t=t_0}^{\infty} \max(b_{ij}(t) - b_i(t), 0))dt \tag{5.6}$$

Der Zeitpunkt t_0 ergibt sich für den Server aus der aktuellen Zeit t_{akt} plus der zu erwartenden Übertragungsdauer d_{ij}, d.h. t_0 beschreibt die Ankunftszeit der Repräsentation r_{ij} auf dem Client. Das Problem der obigen Formel ist die steigende Ungenauigkeit der Abschätzung mit fortlaufender Zeitdauer. Zwar sind b_{ij} und b_i als Funktionen über die Zeit definiert, letztendlich beruhen sie aber auf der Position und dem Blickwinkel des Betrachters. Die Vorhersage dieser Parameter ist jedoch gerade für große t sehr ungewiss, weshalb Integral 5.6 um eine zeitabhängige Gewichtung $w(t)$ erweitert wird. Mit wachsendem t konvergiert $w(t)$ gegen 0, so dass folgendes Integral einen beschränkten Wertebereich hat:

$$\int_{t=t_0}^{\infty} \max(b_{ij}(t) - b_i(t), 0) \cdot w(t - t_{akt})dt \tag{5.7}$$

Die Berechnung dieses Integrales kann in Bezug auf mehrere Elemente und Repräsentationen äußerst kostspielig werden. Schließlich ist es ja das Ziel des Servers, unter den vorhandenen Repräsentationen r_{ij} eines Elements i eine geeignete Darstellung für die Übertragung auszuwählen. Theoretisch kann die Zahl der r_{ij} sogar bis ins Unendliche anwachsen, weil die Imposter-basierten Repräsentationen jedem beliebigen Blickwinkel entsprechen können. Teler et al. versuchen, die Menge der Auswahlmöglichkeiten durch die Vorhersage der Navigation des Betrachters zu limitieren. Hierzu unterscheiden sie zwischen drei verschiedenen Situationen, die während einer Navigation auftreten können, nämlich dem Verharren auf der Stelle, dem Drehen auf der Stelle und gradlinigen Bewegungen. Kurvenverläufe nähern sie über gemittelte gradlinige Bewegungen an. Im Falle des Verharren sind die Funktionen $b_{ij}(t)$ und $b_i(t)$ konstant, weshalb sich Integral 5.7 zu einem Produkt aus $\max(b_{ij}(t) - b_i(t), 0)$ und der Zeitspanne dieser Situation vereinfacht. Hierbei muss erwähnt werden, dass Teler et al. die Gewichtung $w(t)$ nicht über eine allmählich gegen 0 konvergierende Funktion, sondern über eine *Cutoff* Funktion beschreiben. Während der Drehung nehmen Teler et al. vereinfachend an, dass ein Element aus allen Blickwinkeln betrachtet genauso aussieht wie im Zentrum des Bildbereichs. Somit ist das Kriterium der Genauigkeit einer Repräsentation r_{ij} für die Bestimmung der b_{ij} konstant und lediglich das Kriterium der Sichtbarkeit muss in zweifacher Weise integriert werden: Zum einen wenn das Element den Sichtbereich schneidet und zum anderen wenn das Element vollständig im Sichtbereich liegt. Im letzteren Fall ist die Sichtbarkeit konstant. Für ersteren Fall gehen Teler et al. von einer konstanten Drehgeschwindigkeit aus, weshalb die Sichtbarkeit eines Elements linear wächst oder fällt. Das Zeitintervall des Integrales ist über t_0 und t_{dreh} definiert, wobei t_{dreh} sich aufgrund der konstanten Drehgeschwindigkeit leicht über den begrenzenden Blickwinkel der Sichtpyramide bestimmen lässt. Im Falle einer gradlinigen Bewegung berechnen Teler et al. den Korridor, der sich durch das Verschieben der Sichtpyramide entlang der Geraden ergibt. Die Gewichtung $w(t)$ spezifiziert das Ende des Korridors. Während einer gradlinigen Bewegung ändert

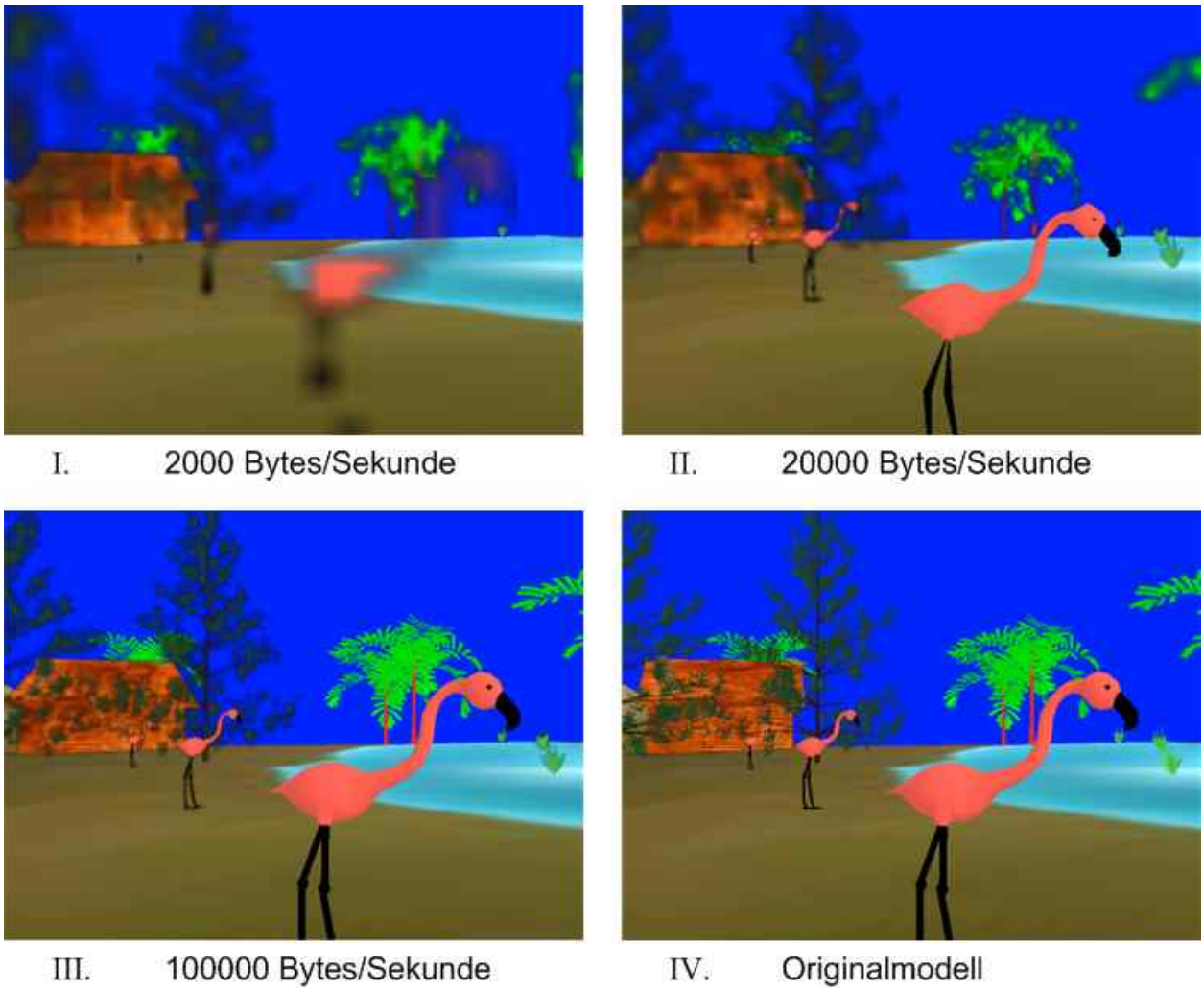

Abbildung 5.10: *Das System von Teler et al. beruht im Wesentlichen auf der Berücksichtigung der Bandbreite. Die einzelnen Bilder setzen die Qualität der Visualisierung bei unterschiedlichen Durchsatzraten im Vergleich zur Originalszene (Screenshot des Systems von Teler et al. [TL01]).*

sich sowohl die Sichtbarkeit als auch die Genauigkeit der Elemente, insofern ist dies der komplexeste der drei Fälle. Teler et al. verwenden daher eine numerische Integration. Das hierzu erforderliche Zeitintervall liegt zwischen dem Zeitpunkt, an dem das Element die Sichtpyramide des Betrachters betritt und dem Zeitpunkt, an dem das Element den Sichtbereich wieder verlässt. Eine weitere Limitierung existiert durch die Übertragungsdauer des Elements und die Gewichtung $w(t)$. Letztere liefert 0 sobald t der zweifachen Dauer entspricht, welche der Benutzer mit der momentanen Schrittgeschwindigkeit für das vollständige Durchschreiten der Sichtpyramide benötigen würde.

Im Gegensatz zu den vorhergehenden Systemen zeichnen sich Teler et al. dadurch aus, auch die visuellen Informationen einer Szene über ein Area-of-Interest Konzept zu selektieren und den gegebenen Leistungsmerkmalen anzupassen. Sie kommen somit den Intentionen der vorliegenden Arbeit zwar sehr nahe, lassen jedoch eine ganze Reihe von Fragen unbeantwortet. Ein grundlegender Punkt ist dabei die These, dass als alleiniges Merkmal die Bandbreite zu berücksichtigen sei. Dies ist bei dem Durchsatz heutiger Netzwerke in Relation zu der Lei-

stungsfähigkeit aktueller mobiler Endgeräte zumindest einer Überprüfung zu unterziehen. Als Beispiel kann ein Szenario dienen, in dem zwar nur ein Polygonnetz aber trotzdem viele Elemente übertragen werden. Der resultierende Transferaufwand fällt relativ gering aus, da neben dem Mesh lediglich die Positionen der Elemente zu versenden sind. Der Visualisierungsaufwand auf Seite des Clients ist dagegen sehr beachtlich. Teler et al. liefern jedenfalls keinen Beweis ihrer These. Die Formeln zur Bestimmung der Genauigkeit 5.1 und 5.2 beinhalten in Form der Höhe beziehungsweise Fläche Informationen, die sich auf die Projektion eines Elements auf den Bildbereich beziehen. Die Berechnung dieser Werte ist selbst unter Zuhilfenahme aktueller Grafik-Hardware nicht gerade billig, da nahezu die komplette Render-Pipeline durchlaufen werden muss. Hierbei darf nicht vergessen werden, dass sämtliche Kalkulationen zu Lasten des Servers gehen. Weiterhin machen Teler et al. keine Angaben wie sie die Werte d_{ij} einer Repräsentation r_{ij} definieren. Weil die Bandbreite aber durchaus starken Schwankungen unterliegen kann, wäre auch an dieser Stelle eine Funktion über die Zeit angebracht. Infolge der Imposter muss der Server grafikfähig sein, was an sich noch kein Hindernis ist. Allerdings erfordert das Rendern der Imposter in Bezug auf mehrere Objekte und Clients einen Rechenaufwand, wie er von einem normalen Standard PC nicht mehr geleistet werden kann. Somit sind als Server nur noch sehr teure Systeme einsetzbar (siehe SGI VizServer). Ein in Hinsicht auf die Imposter vielleicht eher subjektiver Kritikpunkt ist deren inhomogenes Äußeres im Vergleich zu den Polygon-basierten Visualisierungen. Dieser Punkt erhält allerdings Bedeutung wenn das von den Autoren zwar erwähnte aber nicht weiter verfolgte Kriterium der Wechsel der Sichtbarkeit angeführt wird. Obgleich ein großer Teil der Abschätzungen auf einem exakten View Frustum Culling beruht, sind hierzu keine Informationen verfügbar. Entsprechende Verfahren implizieren jedoch selbst mit Unterstützung der Hardware bereits in Bezug auf einen einzelnen Betrachter einen enormen Rechenaufwand. Die im Zusammenhang mit Visibility Culling Verfahren wichtige räumliche Sortierung der Elemente findet ebenso wenig Beachtung wie der Umgang mit dynamischen Szenen. Zusammenfassend kann gesagt werden, dass Teler et al. einen sehr interessanten Ansatz für die Bestimmung der Reihenfolge der zu übertragenden Elemente liefern. Aufgrund des hohen Rechenaufwands zu Lasten des Servers ist allerdings die Anwendbarkeit des Ansatzes in virtuellen Umgebungen in Frage zu stellen.

5.2.6 Weitere Ansätze

Neben den zuvor beschriebenen Konzepten existiert noch eine Reihe anderer Verfahren, die an dieser Stelle kurz erläutert werden. Wie schon in Abschnitt 4.8 angedeutet, lassen sich die Ansätze in Peer-to-Peer und Client-Server Systeme gliedern.

NPSNET[ZPF+93] ist eine virtuelle Peer-to-Peer Umgebung, die auf einer Multicast Verbindung basiert. Es ist für eine große Zahl an Benutzer in militärischen Trainingsszenarien ausgelegt. Jedes aktive Objekt wird durch einen Benutzer kontrolliert, dem sogenannten *Player*. Die Objekte werden auf allen anderen Knoten des Netzwerks als *Ghosts* repliziert. Aus diesem Grund wird diese Vorgehensweise auch als *Player-Ghost Paradigma* bezeichnet. *Bamboo* [WZ98] ist eine Fortführung von NPSNET und erlaubt dynamische Erweiterungen zur Laufzeit. Ein weiterer Multicast Peer-to-Peer Ansatz wurde von Broll [Bro97] vorgestellt. Er verwendet VRML zur Beschreibung der 3D Szenen und verbindet Knoten, die

nicht Multicast fähig sind, über einen Proxy- beziehungsweise Relais-Server. Das System *Scalable Platform for Large Interactive Network Environments* (SPLINE) [WAB+96] ist ein Toolkit zum Erzeugen großer virtueller Umgebungen mit vielen Teilnehmern. Eine SPLINE Welt ist in sogenannte *Locales* unterteilt. Eine bemerkenswerte Applikation mit SPLINE ist *DiamondPark*.

Im Gegensatz zu den oben aufgeführten Konzepten beruhen die folgenden Systeme auf einem Client-Server System. Ein Beispiel ist das *AVIARY* System von Snowdon et al. [SW94]. Einen weiteren Client-Server Ansatz repräsentiert *Virtual Society* von Lea et al. [LHM+97], welches auf einer mit VRML definierten Welt beruht und als Kommunikationsprotokoll das *Virtual Society Client Protocol* (VSCP) verwendet. Das *MR Toolkit* [SGLS93] bietet einen Baukasten für virtuelle Umgebungen. Seine Besonderheit liegt in einer *Shared Memory* Abstraktion. Ein anderes Client-Server System ist *COTERIE* von MaxIntyre et al. [MF96], welches auf einer Replikation der verteilten Objekte und parallelen Programmierkonzepten basiert.

Ein kommerzielles Framework für virtuelle Umgebungen ist das *WorldToolKit* [Sen] von Sense8. Zusammen mit den *World2World* und den *WorldUp* Erweiterungen bietet es eine Entwicklungsumgebung für virtuelle Client-Server Systeme.

5.3 Zusammenfassung

Dieses Kapitel dient der wissenschaftlichen Einordnung der vorliegenden Arbeit sowie der Abgrenzung gegenüber bereits existierenden Ansätzen. Für ersteres wurden zunächst die von der Arbeit betroffenen wissenschaftlichen Bereiche identifiziert und analysiert. Die Untersuchung stellt dabei die Fragen, welche Technologien benötigt werden, welche Technologien schon vorhanden und inwiefern selbige zu verwerten sind. Im Anschluss erfolgte dann eine Diskussion bestehender Gesamtsysteme, deren Intentionen ähnlich zu den in Kapitel 2 aufgestellten Anforderungen ist. Da diese Gesamtsysteme von den zuvor behandelten Techniken Gebrauch machen, obliegen sie den entsprechenden Einschränkungen. Nichtsdestotrotz wurden die jeweiligen Vor- und Nachteile der Systeme erläutert.

Die eingangs erwähnten wissenschaftlichen Bereiche ergeben sich zum Teil direkt aus den in Kapitel 2 eingeführten Anforderungen. Anforderung 2.1 verlangt die Repräsentation großer, dynamischer und interaktiver Szenen und ist durch die Bereiche der Szenen Repräsentation, der Mensch-Maschine-Interaktion sowie der Simulation, Animation und Interaktion abgedeckt. Anforderung 2.2 erwartet die Visualisierung großer, dynamischer und interaktiver 3D Szenen auf verteilten, heterogenen Endgeräten und ist somit neben den bereits genannten Themengebieten vom Bereich der Visualisierung sowie der Übertragung, Kodierung und Kompression betroffen.

Der Bereich der Szenen Repräsentation gliedert sich in mehrere Untergebiete, nämlich in die Adaption und Beschreibung der visuellen und verhaltensspezifischen Informationen eines Elements, dessen plattformübergreifender Darstellung sowie die räumliche Sortierung aller Elemente. Letztere ist beispielsweise für die ausgabesensitive Visualisierung unabdingbar. Die in Abschnitt 4.5 scheiden als Kandidat für die räumliche Sortierung aus, da ihre konsistente Pflege der Kohärenzen in Bezug auf dynamische Szenen kaum aufrecht zu

erhalten ist. Insbesondere erweist sich das Problem der Identifikation benachbarter Elemente als zu aufwändig. Somit bleibt der Ansatz der Raumunterteilungsbäume, für deren Problem der Schnittelemente Lösungsvorschläge von Chang et al. [CLC03] existieren. Auf einen dynamischen Raumunterteilungsbaum sind drei Operationen zu definieren, nämlich das Einfügen eines Elements, das Entfernen eines Elements, sowie die Transformation eines Elements. Jede dieser Operationen kann eine Verletzung der Kapazität δ einer Zelle des Raumunterteilungsbaumes implizieren, wodurch eine aufwändige Reorganisation des Baumes notwendig wird. Das Problem wiegt umso schwerer, weil große Szenen den Speicherplatz der Endgeräte überfordern können und somit eine Out-of-Core Strategie notwendig ist. Sowohl für die effiziente Reorganisation dynamischer Szenen als auch für deren Out-of-Core Behandlung gibt es keine geeigneten Lösungsvorschläge. Sämtliche Out-of-Core Konzepte [AS97, CGL$^+$98, SCH$^+$01, CKS02] sind auf statische Szenen ausgelegt. Aus diesem Grund ist hier der Bedarf nach einem dynamischen Raumunterteilungsbaum mit Out-of-Core Strategie abzuleiten, welcher die drei genannten Operationen bei gleichzeitiger Konsistenz der räumlichen Kohärenzen bereitstellt.

Der objektorientierte Ansatz der Animationselemente ist für die plattformübergreifende Repräsentation eines Elements der Szene nicht ausreichend. Beispielsweise kann ein Animationselement nicht Methoden zur Visualisierung auf beliebigen Betriebssystemen anbieten. An dieser Stelle bietet sich ein komponentenbasiertes Konzept an, wobei jede Plattform eine Anzahl von Komponenten bereitstellt, die bei Bedarf in ein Animationselement eingesetzt werden können. Hierzu ist bezüglich der Animationselemente ein entsprechendes Framework zu spezifizieren, was den transparenten Austausch der Komponenten ermöglicht. Agenten bedeuten in dieser Arbeit eine Erweiterung des komponentenbasierten Programmierparadigmas, weil sie zusätzlich die Beschreibung einer Applikation auf einer aufgabenorientierten und damit abstrakteren Ebene erlauben. Damit ist beispielsweise die Beschreibung von Animationen und Simulationen, also von dynamischen Szenen, auf einer derartigen Ebene möglich. Folglich ist der Begriff der Animationselemente auf die Animationsagenten auszudehnen. Eine äquivalente Bezeichnung existiert bereits in [Dör01], hat aber dort andere Intentionen. Neben der Definition des Frameworks ist zusätzlich ein Konzept für die Beschreibung von aufgabenorientierten Animationen zu erstellen.

Für eine Visualisierung auf heterogenen Endgeräten ist eine adaptive Repräsentation der eigentlichen visuellen Informationen eines Animationsagenten erforderlich. Hierzu sind Level-of-Detail Konzepte zur automatischen Generierung von Detailstufen geeignet. Da die aktuelle Grafik-Hardware ausschließlich Dreiecksnetze berücksichtigt, scheiden Darstellungen mit Hilfe von impliziten Funktionen, parametrischen Funktionen, Multiresolution Analysis oder Unterteilungsflächen aus. Sie bieten zwar bezüglich ihrer Übertragung eine sehr kompakte Beschreibung, müssen aber vor einer Visualisierung aufwändig in ein Dreiecksnetz konvertiert werden. Dies ist insbesondere von mobilen Endgeräten nicht in Echtzeit zu leisten. Resampling Techniken bieten keine genaue Abschätzung der nach der Reduktion verbleibenden Dreiecke. Unglücklicherweise ist dies das entscheidende Kriterium für eine exakte Adaption an die Leistungsmerkmale eines Endgerätes. Folglich reduziert sich die Auswahl möglicher Techniken auf Verfahren zur Geometriereduktion. Hier bieten sich insbesondere progressive Techniken an, da sie die schrittweise Übertragung und Verfeinerung erlauben. Im Gegensatz zu Hoppe [Hop96] und Popovic et al. [PH97] berücksichtigt der Ansatz von

Schröder [Sch97] keine Zusatzattribute wie etwa Normalen oder Texturkoordinaten. Somit verbleiben die beiden erst genannten Ansätze, welche jedoch infolge der Lösung eines Optimierungsproblems sowohl während der Simplifizierung als auch während der Verfeinerung einen hohen Rechenaufwand mit sich bringen. Insofern besteht hier der Bedarf nach einem schnellen Ansatz, welcher nicht nur bei der Reduktion, sondern vor allem bei der Verfeinerung auf schwachen Endgeräten eine geringe Rechenkapazität beanspruchen soll. Weiterhin soll das Verfahren eventuelle Zusatzattribute berücksichtigen, die nach der Simplifizierung in getrennten, progressiven Datenströmen vorliegen sollen. Hierdurch werden einerseits Redundanzen bezüglich der Übertragung vermieden und andererseits eine effiziente Kodierung der Daten durch spezifische Algorithmen ermöglicht. Eine zusätzliche Anforderung ist die simultane Verfeinerung und Visualisierung der Dreiecksnetze auf den Endgeräten, um empfangene Informationen augenblicklich anzeigen zu können.

Der Bereich der Mensch-Maschinen-Interaktion entspricht nicht dem Schwerpunkt der vorliegenden Arbeit. Da aber nach Definition 2.1 die Interaktionsmöglichkeiten zu den Leistungsmerkmalen eines Endgerätes zählen, ist zumindest eine Navigationsschnittelle erforderlich, welche auf allen Endgeräten einschließlich PDAs zum Einsatz kommen kann. Eine Anforderung ist dabei die Gleichberechtigung der Benutzer, ohne Anwender von Systemen mit komfortableren Eingabegeräten zu beschränken. Glücklicherweise existieren bereits bewährte Konzepte, die lediglich auf mobile Endgeräte zu erweitern sind.

In Bezug auf den Bereich der Simulation, Animation und Interaktion ist es erforderlich, für die Beschreibung dynamischer Szenen eine Schnittstelle zu spezifizieren. Zwar sollen keine neuen Animationstechniken entwickelt werden, aber es soll zumindest die Option gegeben sein, mit üblichen Techniken wie Motion Capturing oder inverse Kinematik erstellte Animationen anzuzeigen. Wie schon in Abschnitt 4.8 angedeutet, hat die gewählte Netztopologie entscheidenden Einfluss auf den Kommunikationsaufwand, der mit der Synchronisation dynamischer Vorgänge entsteht. Ziel ist an dieser Stelle nicht nur, den Aufwand während eines dynamischen Vorganges zu reduzieren, sondern bereits zuvor eine skalierte Übertragung der Animationsvorschriften selbst zu erreichen. Die dazu notwendigen Kriterien sollen sich auf die Sichtbarkeit einer Animation beschränken. Mögliche Element-Element-Interaktionen wie etwa Kollisionen sind von einer der Szene zugrunde liegenden Simulation aufzulösen und nicht Thema dieser Arbeit.

Für den Bereich der Visualisierung ist eine ausgabesensitive Bearbeitung von Animationsagenten innerhalb großer Szenen erstrebenswert. Leider berücksichtigen die hierfür geeigneten Occlusion Culling Verfahren in den seltensten Fällen generische, dynamische 3D Szenen. Aufgrund der Ansicht der vorliegenden Arbeit, wonach die Pflege dynamischer Raumunterteilungsbäume in Echtzeit möglich ist, gibt es aber durchaus einige Kandidaten [GKM93, ZMHHI97, BMH99]. Dabei sind vor allem Verfahren von Interesse, welche die gegebene Grafik-Hardware ausnutzen. Der Ansatz von Greene et al. benötigt mit der hierarchischen z-Pyramide eine Form der Hardware, die in dieser Weise niemals ihren Weg in aktuelle Grafikchips gefunden hat. Ohne einen entsprechenden Support ist das Verfahren bezüglich Speicher und Rechenleistung aber zu aufwändig. Der Ansatz von Zhang et al. [ZMHHI97] macht Gebrauch von vorberechneten Occluder-Datenbanken und ist daher in dynamischen Szenen nur bedingt einsetzbar. Das Verfahren von Bartz et al. [BMH99] stellt bei Vorhandensein des HP-Flags oder der NVIDIA-Occlusion Query das derzeit vielleicht

effizienteste Verfahren dar. Jedoch gehören beide Features nicht zum derzeitigen Standard gängiger Grafikbibliotheken wie etwa OpenGL und werden im Falle der NVIDIA Reihe erst ab der GeForce3 unterstützt. Allen Techniken gemeinsam ist das Fehlen einer möglichen Adaption des Rechenaufwands an die Leistungsmerkmale eines Endgerätes, wie es von Anforderung 2.4 verlangt wird. Folglich ergibt sich hier der Bedarf für ein ausgabesensitives Verfahren, welches generische, dynamische 3D Szenen berücksichtigt. Es soll keine Hardware verwenden, die derzeit nicht dem Standard der Grafik-Bibliotheken entspricht und zudem eine Adaption an die Rechenleistung der Endgeräte bieten.

Im Bereich Übertragung, Kodierung und Kompression ist die geeignete Selektion der Daten ein wichtiges Mittel, um eine Adaption der Daten zu erreichen. Grundsätzlich sollten diejenigen Informationen, welche im Interesse des Benutzers liegen, mit einer höheren Priorität zu dem betreffenden Endgerät übertragen werden als solche außerhalb seines Interesses. Hierdurch wird nicht einfach nur die Menge an Daten reduziert, sondern vor allem auch die Reaktionszeit bis der Benutzer ein Feedback erfährt. Somit fällt zunächst eine Spezifikation an, was unter dem Interesse eines Anwenders zu verstehen ist. Für den hierzu korrespondierenden Begriff der Area-of-Interest sollen im Rahmen der vorliegenden Arbeit ausschließlich visuelle beziehungsweise distanzspezifische Kriterien zur Geltung kommen. Eine Möglichkeit besteht darin, lediglich diejenigen Elemente an ein Endgerät zu übertragen, die im Sichtbereich des dortigen Benutzers liegen. Zwar bieten sich dafür Occlusion Culling Verfahren an, allerdings sind diese bezüglich dynamischer Szenen sowie mehrerer Benutzer zu aufwändig. Aus diesem Grund existieren Ansätze [HS98], die das präzise Sichtbarkeitskriterium auf die Distanz zum Betrachter reduzieren. In sämtlichen Fällen gehen die Autoren aber nicht genau auf die Repräsentation der Szene sowie der Identifikation der Animationsagenten innerhalb der Areas-of-Interest ein. Dynamische Szenen finden dort ebenfalls keine Berücksichtigung. Somit ist an dieser Stelle der Bedarf für ein neues Area-of-Interest Konzept festzuhalten. Der auf diesem Konzept basierende Algorithmus zur Identifizierung betroffener Elemente soll mehrere Benutzer in Echtzeit unterstützen, die in einer dynamischen Szene navigieren. Weiterhin soll basierend auf dem Area-of-Interest eine Einstufung der zu übertragenden Elemente nach Dringlichkeit möglich sein.

Aufgrund der begrenzten Bandbreiten ist es erforderlich, die zu übertragenden Datenmengen zu kodieren und gegebenenfalls zu komprimieren. Ein erster Ansatz ist die geforderte Trennung der Datenströme bei der Erzeugung der progressiven Informationen. Verfahren zur Kodierung der Datenströme existieren bereits in vielfacher Zahl und müssen lediglich in das Konzept auf flexible und transparente Weise integrierbar sein.

Mit der Beschreibung des Distributed Interactive Virtual Environments (DIVE) [CH93a, CH93b, Hag96, FS98] beginnt die Diskussion bereits existierender Gesamtsysteme. DIVE hat seinen Ursprung im Jahr 1991 am Swedish Institute of Computer Science. Ziel ist die Bereitstellung eines Frameworks für das Management großer Datenströme, wie sie etwa von virtuellen Umgebungen hervorgerufen werden. DIVE ist nicht ausschließlich auf die Übertragung dreidimensionaler Informationen ausgelegt, sondern integriert den Transfer verschiedener Medien. Zentraler Baustein der DIVE Architektur ist die verteilte Welt-Datenbank, wobei eine DIVE Welt aus einer hierarchischen Anordnung sogenannter Entities besteht. Eine Entity kann einer visuellen, einer akustischen sowie einer verhaltensspezifischen Information entsprechen. Die Repräsentation der 3D Szenen ist über einen IRIS Performer Szenegraphen

realisiert. DIVE basiert auf einem Multicast Peer-to-Peer Netzwerk. Jeder Benutzer kopiert den Ausschnitt der Welt-Datenbank, der in seinem Interesse liegt, auf seinen lokalen Knoten. Verändert ein Benutzer die Welt, so wirken sich die Transformationen zunächst auf die lokale Kopie des Anwenders aus und werden erst im Anschluss an die übrigen Teilnehmer weitergeleitet. Dadurch entstehen kurze Inkonsistenzen bei der Synchronisation dynamischer Vorgänge. Größtes Manko von DIVE ist aber das Fehlen einer Adaption der lokalen Kopie an die Leistungsfähigkeit des jeweiligen Peers. Der verwendete Performer Szenegraph bietet von sich aus weder eine geeignete Organisation dynamischer Szenen noch eine ausgabesensitive Visualisierung. Entsprechende Techniken müssen beziehungsweise können zusätzlich integriert werden.

RING [Fun95] ist ein Client-Server-System für Virtual Environments mit hoher Verdeckungstiefe, welches im Jahr 1995 von Thomas Funkhouser vorgestellt wurde. Funkhousers Intention ist nicht die adaptive Übertragung visueller Informationen, sondern vielmehr der Beweis, dass mit Hilfe visueller Kriterien der Aufwand zur Synchronisation dynamischer Szenen deutlich reduziert werden kann. Als Beispiel verwendet er eine in Zellen unterteilte Innenraumarchitektur, wobei für jede Zelle mittels eines PVS Verfahrens die sichtbaren Nachbarzellen bestimmt werden. Die Szene ist einschließlich eines Standardavatars bereits auf allen Clients vorhanden. Sämtliche Benutzer sind durch den Standardavatar repräsentiert und entsprechen den einzigen dynamischen Elementen der Szene. Betritt ein Avatar eine Zelle, so muss der mit dem Avatar assoziierte Client über alle anderen Avatare informiert werden, die sich im Sichtbereich der Zelle befinden. Aufgrund der hohen Verdeckungstiefe handelt es sich dabei nur um einige wenige Avatare, weshalb ein entsprechend geringer Kommunikationsaufwand entsteht. RING basiert auf einer Server Hierarchie und erlaubt keine direkte Kommunikation unter den Clients. Funkhouser führt hierfür einige wichtige Gründe an. So muss ein Client im Falle seines transformierten Avatars nur eine Nachricht versenden, was einer Entlastung schwacher Endgeräte gleichkommt. Der Server kann die eingehenden Nachrichten filtern und nur an diejenigen Clients weiterleiten, die daran ein Interesse haben. Zusätzlich ist der Server in der Lage, die Nachrichten mit seinem eigenen Wissen zu ergänzen. Da DIVE auf einem PVS Verfahren beruht, sind seine Anwendungsmöglichkeiten beschränkt. Wichtig ist hier die Reduktion des Nachrichtenaufwands durch visuelle Kriterien. Dieses Konzept lässt sich auch auf die visuellen Informationen einer Szene erweitern.

Virtual Reality Modeling Language (VRML) [Con] ist eine bekannte Sprache zur Modellierung virtueller Welten, deren Wurzeln in das Jahr 1994 zurückreichen. Ähnlich zu DIVE liegt auch hier die Intention im Transfer verschiedener Medien, wobei der 3D Bereich im Wesentlichen auf dem ASCII-basierten Open Inventor Format [SGIb] formiert. Aufgrund der Unterstützung von Web Inhalten, weist VRML ebenfalls einige Parallelen zur HyperText Markup Language (HTML) auf. Im Jahre 1996 wurde die VRML 2.0 Spezifikation zu einem de facto Standard für die Übertragung von 3D Informationen. Später ging das VRML Konsortium in das heutige Web3D Konsortium über. Ein VRML Szenegraph besteht aus mehreren Knoten, die beispielsweise das visuelle Erscheinungsbild der Elemente einer Szene beschreiben. Andere Knoten dienen als Ereignisquellen beziehungsweise als Ereignissenken, um dynamische Vorgänge zu definieren. Das VRML Prinzip beruht auf einem klassischem Client-Server System. Die Clients laden sich eine Szene vom Server herunter und können selbige mittels eines Browsers betrachten. Letztlich hat sich VRML nicht wirklich als Standard

durchsetzen können, da es keine adaptive und selektive Übertragung einer Szene erlaubt. Aus diesem Grund ist die Grafikqualität großer VRML Szenen äußerst bescheiden. Ebenfalls bietet VRML keine Lösungen für die Organisation dynamischer Szenen sowie deren ausgabesensitiver Visualisierung. Für den neuen Standard X3D ist eine Beurteilung noch zu früh, wenngleich seine 3D Anbindung im Wesentlichen auf dem folgenden MPEG-4 beruht.

MPEG steht stellvertretend für Moving Picture Expert Group. Wie MPEG-1 und MPEG-2 war auch MPEG-4 eigentlich als Format zur Komprimierung von Filmen, d.h. zur Kodierung von Video- und Audiodaten vorgesehen [PE98]. Aufgrund des wachsenden Bedarfs an der Übertragung multimedialer Daten über das Internet wurde die Zielsetzung von MPEG-4 aber um den Transfer von Audio- und Videosequenzen, Texten sowie zweidimensionaler und dreidimensionaler Grafik erweitert. Letzteres ist in Form des Binary Format For Scenes (BIFS) berücksichtigt, wobei BIFS abgesehen von geringen Ergänzungen einem binären VRML 2.0 Derivat entspricht. Hierdurch ist eine bedeutend höhere Kompression der Daten gegeben, jedoch geht auch die Option verloren, die Szenen mit einem beliebigen Texteditor zu bearbeiten. Analog zu VRML beruht MPEG-4 ebenfalls auf einer Client-Server Struktur. MPEG-4 verfügt über ein flexibles Schichtenmodell, welche eine Transparenz hinsichtlich des zugrunde liegenden Netzwerkes ermöglicht. Zudem bietet es die Übertragung getrennter Datenströme sowie deren Synchronisation auf empfangender Seite. Grundsätzlich erbt MPEG-4 sämtliche Schwächen von VRML und bietet darüber hinaus keine Spezifikation, wie eine Visualisierung einer MPEG-4 zu realisieren ist.

Ebenso wie bei der vorliegenden Arbeit liegt die Intention des Client-Server Systems von Teler et al. in der adaptiven und selektiven Übertragung von 3D Szenen. Ihrer These nach sind nicht die Leistungsmerkmale der Endgeräte das entscheidende Kriterium, sondern die Bandbreite der verbindenden Netzwerke. Einen Beleg dieser These bleiben sie allerdings schuldig. Teler et al. versuchen, eine Adaption der Daten nicht nur durch die Reduktion der geometrischen Informationen eines Elements zu erreichen, sondern auch durch die Bereitstellung verschiedener Repräsentationen. Beispielsweise existiert neben der geometrischen Repräsentation eines Elements immer die Möglichkeit, selbiges durch einen Imposter zu beschreiben, was einer 2D Textur gleichkommt. Imposter sind nicht invariant gegenüber Transformationen, obgleich es nach Teler et al. Toleranzbereiche gibt. Die Absicht von Teler et al. beruht nun darauf, für alle im Sichtbereich eines Betrachters liegenden Elemente eine ideale Repräsentation mit bestmöglicher Genauigkeit unter Berücksichtigung der Bandbreite zu übertragen. Dieses Problem dehnen sie auf die interaktive Navigation des Betrachters aus, weshalb sie eine Optimierung hinsichtlich des prekalkulierten Navigationspfades anstreben. Aufgrund der Imposter Technik benötigen Teler et al. einen grafikfähigen Server, der durch die hohe Belastung bei mehreren Clients keinem Standard PC mehr entsprechen kann. Teler et al. gehen von einer konstanten Bandbreite aus und geben zudem keine genauen Angaben zur Repräsentation der Szene. Für die Identifikation der Areas-of-Interest verwenden sie ein hierarchisches View Frustum Culling, was aber in Bezug auf mehrere Clients zu aufwändig ist. Gleiches gilt auch für die Berechnung der angegebenen Formeln, welche zum Teil Informationen über die Projektion der Elemente auf die Bildfläche verwenden.

Kapitel 6

Konzept

Dieses Kapitel stellt ein Konzept vor, welches sowohl den grundlegenden Anforderungen in Kapitel 2 als auch den in Kapitel 5 spezifizierten Techniken genügt. Anhand der dort aufgeführten Kriterien wird zunächst ein abstraktes Modell der Architektur entworfen und dieses dann schrittweise verfeinert.

6.1 Ein aufgabenorientiertes Modell

Wie in Abschnitt 4.8 zu erkennen, gilt es eine grundlegende Entscheidung zwischen einer Peer-to-Peer oder einer Client-Server Topologie zu treffen. Erstere hat wie etwa im Falle von DIVE (siehe Abschnitt 5.2.1) den Vorteil, für eine gleichmäßige Auslastung der Endgeräte zu sorgen. Zudem ist sie weitestgehend resistent gegen den Ausfall einzelner Knoten, da die Systeme nicht von der Bereitschaft eines einzelnen Gerätes oder einiger weniger Geräte abhängen. Jedoch sind Peer-to-Peer Konzepte in Bezug auf dynamische Vorgänge entweder nur mit Einschränkungen oder lediglich sehr umständlich zu synchronisieren. Auch hier kann wiederum DIVE als Beispiel herangezogen werden: Unter Inkaufnahme kurzfristiger Inkonsistenzen wirken sich Manipulationen erst auf die lokale Kopie des Verursachers aus und werden dann zu den anderen Teilnehmern propagiert. Im Bereich der dynamischen Szenen haben Client-Server Ansätze einen bedeutenden Vorteil, weil eine zentrale, konsistente Repräsentation der Szene auf dem Server existiert. Außerdem kann, wie von Funkhouser [Fun95] gezeigt, in einer Topologie mit N Knoten die Anzahl der Nachrichten infolge eventueller Veränderungen deutlich unter $O(N)$ liegen. Abschnitt 5.2.2 beschreibt einige weitere Vorteile eines Client-Server-Systems. Ein Nachteil derartiger Konzepte liegt oftmals in der hohen Belastung der Server, manchmal allein schon durch die Nachrichtenverarbeitung bedingt. Aufgrund des erklärten Ziels der vorliegenden Arbeit, eine auch für Standard PCs taugliche Lösung anzubieten, ist dieser Aspekt nicht zu vernachlässigen. Da aber ein weiterer Schwerpunkt auf der Berücksichtigung dynamischer 3D Szenen beruht, gibt letztlich das Argument der einfacheren Synchronisation den Ausschlag für ein Client-Server-Konzept. Zudem zeigen die in Abschnitt 4.8 erwähnten Arbeiten von Funkhouser, dass mit Hilfe hierarchischer Server-Strukturen durchaus eine Entlastung des Servers beziehungsweise eine effiziente Verteilung der anfallenden Arbeiten auf eine Gruppe von Servern möglich ist. Die

Option mehrerer Server oder gar eines Server-Clusters darf aber nicht als Entschuldigung dienen, einen Server über eine unzumutbare Menge an Aufgaben seiner Handlungsfähigkeit zu berauben. Vielmehr ist die Intention, ein in der Praxis umsetzbares Konzept zu entwerfen, welches als Server einen einzelnen Standard PC verwendet. Eine wichtige zu klärende Frage ist dabei die Rollenverteilung zwischen Server und Client, d.h. welche Aufgaben obliegen dem Server und welche dem Client.

Der Schwerpunkt des Servers besteht in der Bereitstellung einer oder mehrerer 3D Szenen. Hierzu müssen dem Server entsprechende Beschreibungen etwa in Form von Dateien zugänglich sein. Wünschenswert ist die Unterstützung verschiedener Dateiformate, d.h. der Server muss in der Lage sein, neben einem eigenen Format auch bereits existierende Formate zur Beschreibung von 3D Szenen wie etwa VRML einzuladen und zu verarbeiten. Nach dem Einlesen müssen die Daten der Szene in einer geeigneten Form repräsentiert werden. In diesem Zusammenhang sei insbesondere an die konsistente Darstellung dynamischer Szenen erinnert, deren Simulation ebenfalls dem Server zuzuordnen ist. Der Einfluss der Clients beschränkt sich hier auf eventuelle Interaktionen des Benutzers. Bezüglich der Repräsentation einer Szene existieren auf Seite des Servers aber noch andere wichtige Anforderungen: Zum einen die Identifikation einzelner Bereiche der Szene entsprechend eines Area-of-Interest Konzepts und zum anderen die interaktive Erzeugung eines progressiven Formats aufgrund möglicher Modifizierung eines Elements der Szene. Neben der Selektion der Daten ist es die Entscheidung des Servers, welche Menge und welche Art an Daten ein Client erhält. Dazu benötigt der Server Informationen über die Leistungsfähigkeit der Clients, weshalb er den Clients eine Schnittstelle zur Registrierung und Deregistrierung anbieten muss. Weiterhin ist die Möglichkeit des Nachrichtenaustauschs zwischen Client und Server in mehrerer Hinsicht erforderlich, beispielsweise zum Zwecke der Übertragung visueller Informationen oder zum Zwecke der Synchronisation dynamischer Vorgänge. Aufgrund der begrenzten Bandbreite der Netzwerke benötigt der Server Hilfsmittel zur Kodierung und Kompression der Informationen ebenso wie zur Dekodierung und Dekompression.

Im Gegensatz zum Server liegt die primäre Intention des Clients in der Interaktion mit dem Benutzer. Ein wichtiger Bestandteil ist dabei die Visualisierung der eingehenden Daten, d.h. der Client muss in der Lage sein, die Daten einer Szene zu empfangen, zu repräsentieren und wenn möglich ausgabesensitiv anzuzeigen. Eine Visualisierung auf Seiten des Servers macht allenfalls in Bezug auf Test- oder Kontrollbeobachtungen Sinn. Hierzu wird aber in der Regel keine visuell hochwertige Anzeige, sondern vielmehr ein dem Zwecke der Kontrolle entsprechendes Level-of-Abstraction verlangt (siehe Abschnitt 5.1.4). Zwischen Empfang und Repräsentation der Daten ist durch den Client noch einiges an Arbeit zu leisten. So muss er die empfangenen Informationen dekodieren und dekomprimieren sowie die einzelnen Datenströme eines Elements synchronisieren. Hinsichtlich des letzten Aspekts sei an die in Kapitel 5.1.1 geforderte Auftrennung der Datenströme erinnert. Im Falle progressiver Information ist zudem das Einfügen in die bereits vorhandenen Strukturen vonnöten. Um den Datenaustausch aufgrund dynamischer Vorgänge so minimal wie möglich zu halten, muss der Client in der Lage sein, einfache Animationen innerhalb eines bestimmten Zeitintervalls selbst zu berechnen und umzusetzen.

Anhand der in den beiden letzten Absätzen beschriebenen Aufgaben lässt sich ein erstes Modell des Client-Server-Systems ableiten, wie es in Abbildung 6.1 illustriert ist. Eine zentrale

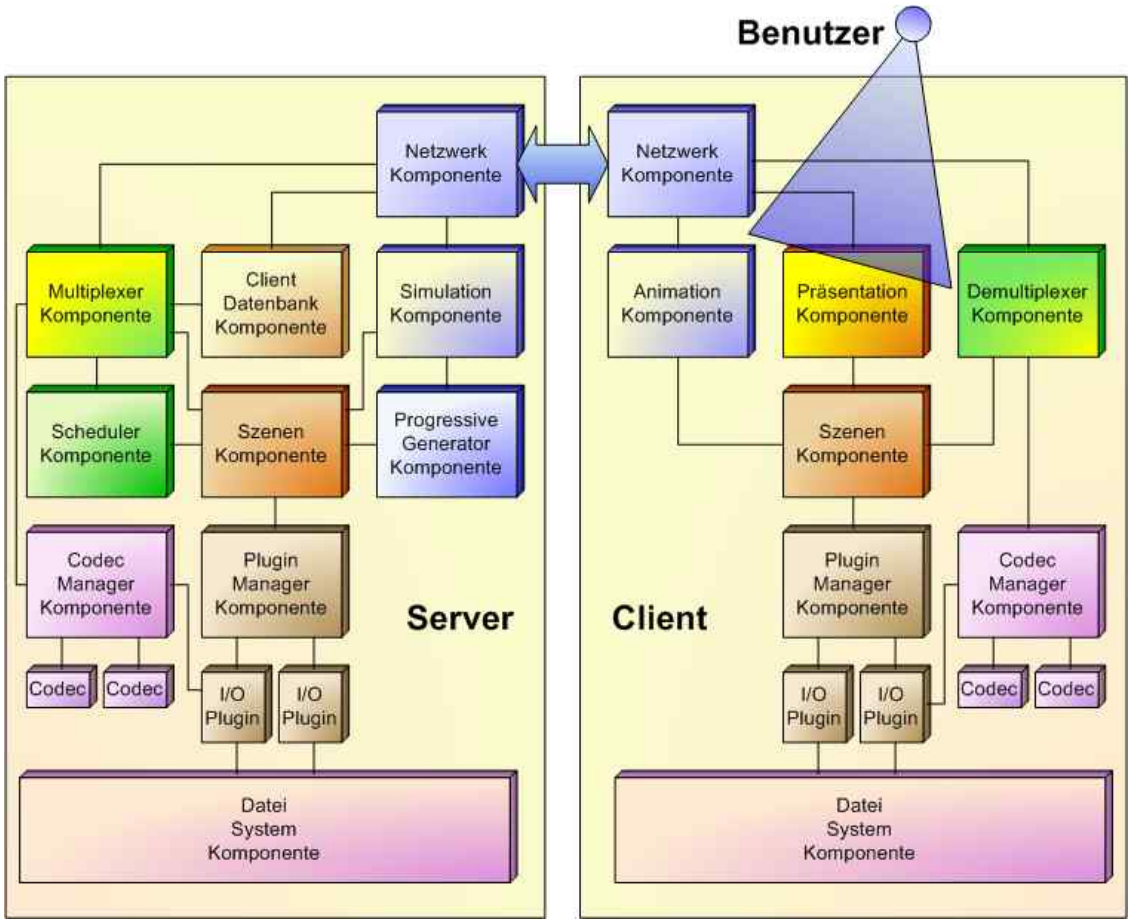

Abbildung 6.1: *Ein erstes aufgabenorientiertes Modell, in dem die Repräsentation der Szene eine zentrale Rolle spielt. Der Benutzer kommuniziert mit der Präsentation-Komponente des Clients.*

Rolle innerhalb des Modells nimmt die *Szenen-Komponente* ein. Sie dient der Repräsentation der großen, dynamischen und interaktiven Szenen. Eine entsprechende Komponente muss auf Server und Client vorhanden sein, wenn auch mit unterschiedlichen Anforderungen. Beispielsweise benötigt Funkhouser[Fun95] für die Darstellung der Szene auf Seite des Servers keine visuellen, sondern lediglich räumliche Informationen. Allerdings geht er auch davon aus, dass die erforderlichen Daten bereits auf den Clients vorhanden sind. Ein derartiges Szenario ist hier also nicht denkbar, da ja ein Ziel in der adaptiven Übertragung der visuellen Daten besteht. Wichtig ist aber die Erkenntnis, dass eine Komponente hinsichtlich Server und Client zwar auf einer abstrakten Ebene eine ähnliche oder gar äquivalente Aufgabe haben kann aber trotzdem unterschiedlichen Ansprüchen genügen muss. Um welche Anforderungen es sich dabei handelt, hängt von den übrigen Bausteinen des Modells ab, die von fraglicher Komponente Gebrauch machen. Andere Komponenten wiederum bleiben nahezu unbehelligt von den differierenden Kriterien für Client und Server. Als Beispiele hierzu können die *Plugin-Manager-Komponente* und die *Codec-Manager-Komponente* dienen. Erstere verwaltet sogenannte I/O-Plugins, welche jeweils ein spezifisches Dateiformat in die Szenen-Komponente einlesen beziehungsweise aus der Szenen-Komponente herausschreiben. Beispielsweise könnte die Option eines DIVE, VRML oder X3D Plugins bestehen. Die Aufgabe der Plugin-Manager-Komponente ist die Auswahl und schnelle Bereitstellung

des geeigneten I/O Plugins. Für die Verarbeitung der einzelnen Dateiformate bedienen sich die I/O Plugins der *Codec-Manager-Komponente*. Deren Funktionalität liegt ähnlich der Plugin-Manager-Komponente in der Verwaltung verschiedener Codecs. Jeder Codec ist in der Lage, einen bestimmten Datentyp effizient zu kodieren beziehungsweise zu dekodieren. Zum Beispiel könnte ein Edgebreaker Codec [Ros99] für die Komprimierung der Polygonnetze verwendet werden. Die Begriffe Dateiformat und Datentyp sind an dieser Stelle also nicht synonym zu verstehen. Vielmehr können innerhalb eines Dateiformats verschiedene Codecs zu dessen Verarbeitung zum Einsatz kommen. Prinzipiell ist eine Plugin-Manager-Komponente im Falle des Clients nicht unbedingt erforderlich. Warum soll der Benutzer lokal eine VRML Datei laden können, wenn er die entsprechenden Informationen doch durch den Server erhält? Mit dem Vorhandensein der Plugin-Manager-Komponente erhöht sich aber die Flexibilität des Clients, weil er so auch als autonome Lösung einsetzbar ist. Darüber hinaus bietet sich dem Benutzer die Option, von den empfangenen Daten eine lokale Kopie anzulegen und diese später in aller Ruhe zu betrachten oder zu bearbeiten. Wie in Abbildung 6.1 dargestellt, erfolgt mit Hilfe der *Datei-System-Komponente* eine gesonderte Kapselung des zugrunde liegenden Dateisystems. Eine derartige Abgrenzung ist äußerst wichtig und zwar aus zwei Gründen: Zum einen ist selbst der Speicherplatz einer Festplatte begrenzt, weshalb eine zentrale Kontrollinstanz zur Überwachung von Einfüge- beziehungsweise Ausfügeoperationen erforderlich ist. Zum anderen gibt es mit den PDAs Endgeräte, die in dem Sinne gar nicht über ein Dateisystem beziehungsweise über einen physikalischen Datenträger verfügen. Es wäre beispielsweise fatal ein Out-of-Core Rendering zu realisieren, welches auf einem nicht vorhandenem Dateisystem fußt. Da zusätzlich in bestimmten Fällen Unterschiede zwischen den einzelnen Betriebssystemen auftreten, offeriert die Datei-System-Komponente eine Absicherung gegen derlei Tücken.

Neben der Plugin-Manager-Komponente ist die *Scheduler-Komponente* auf Seite des Servers eine weitere Komponente, die von der Repräsentation der Szene Gebrauch macht. Ihre Ausgabe ist die Selektion der an einen Client zu übertragenden Elemente einer Szene. Dazu benötigt die Scheduler-Komponente Informationen über die aktuelle Area-of-Interest eines Clients. Die Verwaltung der Client-spezifischen Daten wie etwa die besagte Area-of-interest oder die jeweiligen Leistungsmerkmale obliegt der *Client-Datenbank-Komponente*. Nach der Identifizierung der für den betreffenden Client interessanten Elemente übergibt die Scheduler-Komponente die gewonnenen Erkenntnisse der *Multiplexer-Komponente*. Für jedes selektierte Element ermittelt die Multiplexer-Komponente die damit zusammenhängenden Datenströme wie etwa Scheitelpunkte, Normalen oder Texturkoordinaten. Weiterhin bestimmt sie mit Hilfe der in der Client-Datenbank-Komponente eingetragenen Leistungsmerkmale des betroffenen Clients sowohl die Art als auch die Länge der Datenströme. Für die Kodierung und Komprimierung der anfallenden Datenströme bedient sich die Multiplexer-Komponente der Codec-Manager-Komponente. Anschließend übergibt die Multiplexer-Komponente die Datenströme der *Netzwerk-Komponente* zur Übertragung. Letztere verfügt über keine besondere Intelligenz. Ihre Intention ist unabhängig von Server oder Client der effiziente Austausch von Daten zwischen den beiden Instanzen. Nebenbei kapselt sie ähnlich der Transport Layer von MPEG-4 (siehe Abschnitt 5.2.4) die dem Netzwerk zugrunde liegenden Übertragungsprotokolle. Die *Demultiplexer-Komponente* auf Seite des Clients stellt den Gegenpol zur Multiplexer-Komponente dar. Sie synchronisiert die von der Netzwerk-Komponente eingehenden Datenströme, wobei sie zum Zwecke der Dekodierung und Dekomprimierung auf

die Cocec-Manager-Komponente zurückgreift. Ist die Reproduktion eines Elements soweit fortgeschritten, dass eine Visualisierung desselben möglich ist, so trägt die Demultiplexer-Komponente das Element in die Repräsentation der Szene beziehungsweise in die Szenen-Komponente ein.

Zwei miteinander kooperierende Komponenten sind die *Simulation-Komponente* auf Seite des Servers und die *Animation-Komponenten* auf Seite des Clients. Ihre Aufgaben liegen in der Berechnung und Synchronisation dynamischer Vorgänge. Während die Simulation-Komponente die komplette Simulation einschließlich möglicher Benutzerinteraktionen berücksichtigt, ist die Animation-Komponente lediglich dazu imstande, ihr von der Simulation-Komponente übermittelte Ereignisse auf die betroffenen Elemente anzuwenden. Üblicherweise handelt es sich dabei um die Auswahl einer bereits auf dem Client vorhandenen Animationsvorschrift, die mit Hilfe der erhaltenen Werte parametrisiert wird. Im Zusammenhang mit der Simulation-Komponente ist eine weitere Komponente des Servers zu nennen, nämlich die *Progressive-Generator-Komponente*. Ihr fällt die in Abschnitt 5.1.1 spezifizierte Aufgabe zu, ein Element während der Laufzeit über ein sehr schnelles Reduktionsverfahren in ein progressives Format umzuwandeln. Eine Konzeption dieser Komponente als eigenständiger Baustein ist aber ebenfalls sinnvoll. Hierdurch ist es möglich, Elemente bereits in einem Vorverarbeitungschritt zu konvertieren, als Datei abzuspeichern und später dann mit Hilfe eines geeigneten I/O Plugins einzulesen. Die Elemente selbst rekrutieren dabei aus einem Designprozess mit Hilfe eines entsprechenden Werkzeugs wie etwa 3D Studio Max [3DS] oder Rhinoceros [Ass].

Entsprechend dem Schwerpunkt des Clients existiert dort eine *Präsentation-Komponente*, welche demselben Zweck dient wie der gleichnamige Baustein eines VRML-Browsers (siehe Abschnitt 5.2.3). Sie bietet dem Benutzer eine grafische Bedienungsoberfläche, über welche die Kommunikation zwischen Mensch und Maschine stattfindet: Der Benutzer erhält eine Visualisierung der Szene und ist unter Zuhilfenahme verschiedener Eingabegeräte in der Lage, mit den Elementen der Szene und infolgedessen auch mit anderen Benutzern zu interagieren. Gleichzeitig kapselt die Präsentation-Komponente mögliche Abhängigkeiten der Oberfläche vom zugrunde liegenden Betriebssystem sowie das verwendete Interaktionsmodell. Eine andere Aufgabe der Präsentation-Komponente liegt in der Fütterung der Client-Datenbank-Komponente auf Seite des Servers mit Daten. Hierzu zählen einerseits die aktuelle Area-of-Interest des Benutzers und andererseits die Leistungsmerkmale des Clients. Im Falle der Area-of-Interest ist die Verantwortung der Präsentation-Komponente naheliegend, da sie die Sichtparameter des Betrachters anhand der Benutzerinteraktionen nachvollziehen kann. Anders sieht es bei den Leistungsmerkmalen aus, die ja eigentlich clientspezifische Daten darstellen und deshalb nicht unbedingt der Präsentation-Komponente zuzuordnen sind. Eine automatische Bestimmung dieser Leistungsmerkmale ist zwar möglich, sollte aber dem Benutzer für sein Einverständnis vorgelegt werden. An dieser Stelle kommt dann die Präsentation-Komponente ins Spiel.

Die folgenden Tabellen fassen noch einmal alle Aufgaben der einzelnen Komponenten zusammen. Sie sind dreigeteilt in gemeinsame, Server-spezifische und Client-spezifische Komponenten.

Komponente	Aufgabe
Szenen-Komponente	Repräsentation einer großen, dynamischen und interaktiven 3D Szene.
Plugin-Manager-Komponente	Verwaltung der vorhandenen I/O Plugins. Jedes I/O Plugin kann ein bestimmtes Dateiformat lesen beziehungsweise schreiben.
Codec-Manager-Komponente	Verwaltung der vorhandenen Codecs. Jeder Codec kann einen bestimmten Datentyp effizient kodieren beziehungsweise dekodieren.
Netzwerk-Komponente	Austausch von Daten sowie Kapselung der Netzwerkprotokolle.
Datei-System-Komponente	Kontrolle und Kapselung des eventuell plattformabhängigen Dateisystems.

Tabelle 6.1: *Die Aufgaben der hinsichtlich Client und Server gemeinsamen Komponenten.*

Komponente	Aufgabe
Scheduler-Komponente	Selektion der Elemente, die innerhalb der Area-of-Interest eines Clients liegen.
Multiplexer-Komponente	Identifizierung und Parametrisierung der mit einem selektierten Element anfallenden Datenströme bezüglich Länge und Typ.
Client-Datenbank-Komponente	Verwaltung Client-spezifischer Daten wie etwa die Leistungsmerkmale oder die Area-of-Interest.
Simulation-Komponente	Berechnung und Synchronisation der Simulation oder Animation.
Progressive-Generator-Komponente	Konvertierung eines Elements in ein progressives Format zur Laufzeit.

Tabelle 6.2: *Die Aufgaben der Server-spezifischen Komponenten.*

Komponente	Aufgabe
Demultiplexer-Komponente	Synchronisation der eingehenden Datenströme und Reproduktion der betroffenen Elemente.
Animation-Komponente	Starten und Kontrollieren einer auf dem Client vorhandenen Animationsvorschrift für ein bestimmtes Element.
Präsentation-Komponente	Visualisierung und Interaktion mit dem Benutzer sowie Kapselung der Oberfläche und des Interaktionsmodells von der zugrunde liegenden Plattform. Fütterung der Client-Datenbank-Komponente mit Daten

Tabelle 6.3: *Die Aufgaben der Client-spezifischen Komponenten.*

6.2 Ein kommunikationsorientiertes Modell

Während Abschnitt 6.1 sich mit den Aufgaben der einzelnen Komponenten auseinandersetzt, behandelt dieser Abschnitt die Kommunikation zwischen den Komponenten. Einige Interaktionen der Komponenten lassen sich bereits aus der vorangehenden Beschreibung der Aufgaben ableiten, da eine entsprechende Analyse ohne einen gewissen komponentenübergreifenden Kontext nicht sehr aussagekräftig wäre. An dieser Stelle wird nun genau geklärt, warum eine Komponente mit einer anderen kommuniziert und in welcher Form. Hierbei sind zwei verschiedene Aspekte zu berücksichtigen, zum einen die Datenklasse der Kommunikation und zum anderen die Art der Kommunikation. Erstere identifiziert, ob es sich bei den kommunizierten Informationen um reine Daten oder um Befehle wie etwa Kontroll- oder Synchronisationskommandos handelt. Eine genaue Definition, was nun wirklich als Daten oder Befehle zu werten ist, fällt dabei schwer, weil in der Regel auch Befehle nicht ohne eine gewisse Menge an Informationen auskommen. Aus diesem Grund wird oftmals die Menge der Informationen als Kriterium verwendet und folglich ab einem bestimmten Aufwand das Übergeben eines Befehls mit dem Austausch von Daten gleichgesetzt. Die Art oder der Typ einer Kommunikation gibt an, ob es sich um einen lesenden oder einen schreibenden Zugriff handelt. Einige der Zugriffe sind geräteübergreifend, d.h. eine Komponente des Servers kommuniziert mit einer anderen Komponente auf Seiten des Clients oder umgekehrt. In diesem Fall wird davon ausgegangen, dass die Netzwerk-Komponente eine metaphorische Brücke schlagen kann und eine in Bezug auf das zugrunde liegende Netzwerk transparente Kommunikation ermöglicht. Sie bleibt daher in entsprechenden Fällen außen vor.

Abbildung 6.2 illustriert einen von Client und Server unabhängigen Vorgang, wie er beim Laden oder Speichern einer Szene auftritt. Auslösende Kraft ist dabei eine in Abbildung 6.1 nicht näher spezifizierte Kontrollinstanz. Hinter dieser verbirgt sich entweder der Benutzer eines Clients, der mit Hilfe der Präsentation-Komponente die Bearbeitung einer Datei verlangt oder ein Administrator, welcher über ein entsprechendes Tool den Server direkt bedient. Im Falle eines hierarchischen Server-Modells kann es sich auch um einen anderen Server handeln. Zu Beginn des Vorgangs wendet sich die Kontrollinstanz an die Plugin-Manager-Komponente und ordert ein für das Format geeignetes I/O Plugin. Nach Empfang von selbigem bittet die Kontrollinstanz das betreffende Plugin, die gewünschte Datei zu lesen oder zu speichern. Daraufhin bemüht das Plugin die Datei-System-Komponente um den Zugriff auf die Datei in Form eines Handle. Sobald das Handle verfügbar ist, beginnt das Plugin mit dem eigentlichem Datentransfer. Hierzu werden während eines Lesevorgangs die Daten aus der Datei in die Szenen-Komponente übertragen. Im umgekehrten Fall eines Schreibvorgangs konserviert das I/O Plugin die Informationen aus der Szenen-Komponente in der Datei. In bestimmten Situationen kann das Kodieren oder Dekodieren der Daten erforderlich sein, weshalb das I/O Plugin von der Codec-Manager-Komponente einen für den jeweiligen Datentyp geeigneten Codec verlangt. Ist ein solcher Codec vorhanden, so übergibt das I/O Plugin dem Codec die zu verarbeitenden Daten und erhält das Resultat zurück.

Es existiert noch ein anderer Part des Lade- oder Speichervorgangs, der nicht in Abbildung 6.2 vermerkt ist, nämlich der Umgang mit simulationsspezifischen Informationen. Diese können entweder in die Beschreibung der Szene integriert sein oder in einem gesonderten Vorgang geladen oder gespeichert werden. Nach dem vollständigen Laden einer Szene

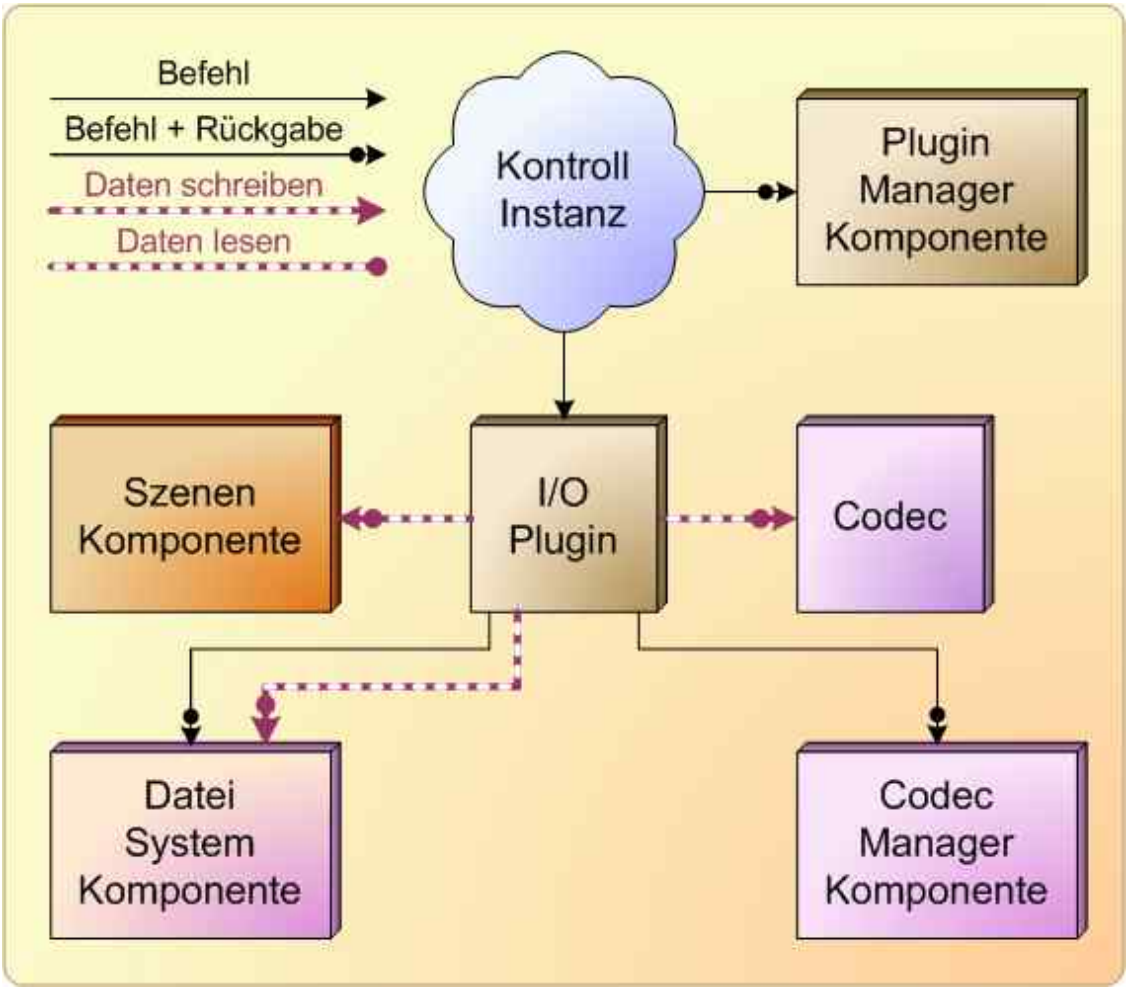

Abbildung 6.2: *Das Laden beziehungsweise Speichern einer Szene wird durch eine Kontrollinstanz ausgelöst, die entweder dem Benutzer eines Clients oder dem Administrator des Servers entspricht.*

einschließlich der Simulation befindet sich der Satz an Animationsvorschriften, welcher während der Simulation verwendet wird, ebenso wie die visuellen Informationen in der Szenen-Komponente.

In Abbildung 6.2 ist die Unterscheidung zwischen Befehlen und Daten relativ einfach, zum einen weil letztere ausschließlich Informationen der Szene repräsentieren und zum anderen weil die Kommandos nur auf wenige Informationen wie etwa dem Dateinamen angewiesen sind.

Abbildung 6.3 beschreibt das Übertragen einer Szene zu einem Client nach dessen Registrierung beim Server und dem dortigen Laden der Szene. Entsprechend handelt es sich hier um einen geräteübergreifenden Vorgang. Der besseren Übersicht wegen sind die Szenen-Komponente und die Codec-Manager-Komponente von Client und Server zu jeweils einem Baustein zusammengefasst. In dem dargestellten Szenario ist die Scheduler-Komponente die aktive Instanz. Sie liest aus der Client-Datenbank-Komponente die Informationen über die Area-of-Interest des Clients. Anhand der Area-of-Interest und der aus der Szenen-Komponente erhältlichen Daten identifiziert und markiert die Scheduler-Komponente die betroffenen Ele-

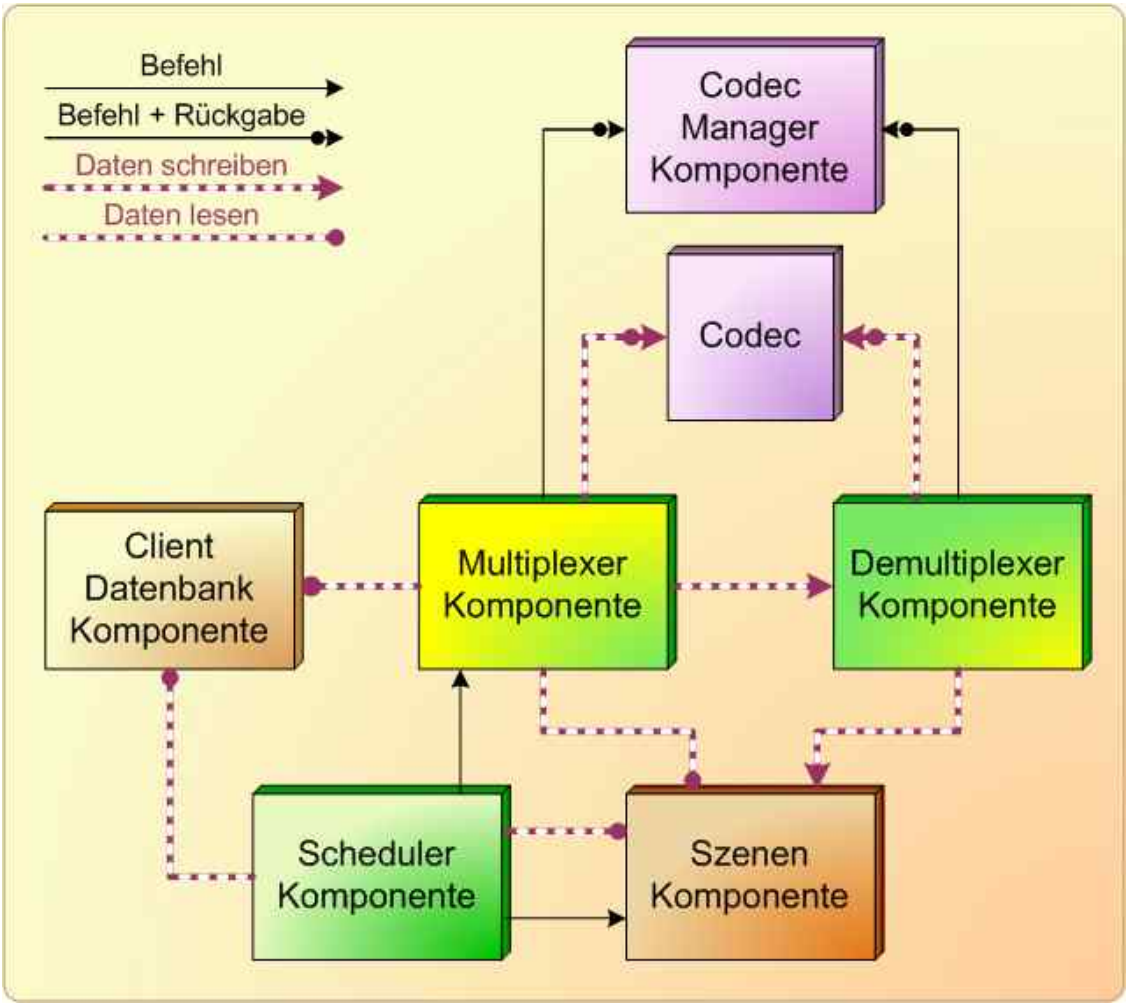

Abbildung 6.3: *Die Scheduler-Komponente selektiert die zu übertragenden Elemente einer Szene und übergibt sie der Multiplexer-Komponente. Diese entscheidet mit Hilfe der Client-Datenbank-Komponente die anfallende Datenmenge und kodiert selbige durch spezifische Codecs. Anschließend übermittelt sie die Informationen der Demultiplexer-Komponente des Clients.*

mente der Szene [1]. Sie übergibt der Multiplexer-Komponente die Referenzen der selektierten Elemente mit der Aufforderung, die Elemente an den Client zu übertragen. Die Multiplexer-Komponente analysiert zunächst mit Hilfe der Informationen aus der Szenen-Komponente, welche Datenströme mit den Elementen verbunden sind. Weiterhin bestimmt sie über die in der Client-Datenbank-Komponente eingetragenen Leistungsmerkmale, welche Datenströme letztendlich für den Client in Frage kommen und wieviele Informationen diese maximal beinhalten dürfen. Im Falle dynamischer Szenen kann es sich bei den anfallenden Daten auch um Animationsvorschriften handeln. Zur Kodierung wendet sich die Multiplexer-Komponente an die Codec-Manager-Komponente und erbittet für jeden Datenstrom einen geeigneten Codec, den sie mit der Komprimierung der Informationen beauftragt. Danach übermittelt sie die kodierten Datenströme der Demultiplexer-Komponente auf Seite des Clients. Ana-

[1]Das Markieren der selektierten Elemente ist der Grund für die Kommandoverbindung zwischen Scheduler-Komponente und Szenen-Komponente. Der Sinn hinter dem Markieren wird später im Konzept des out-of-core Renderns deutlich.

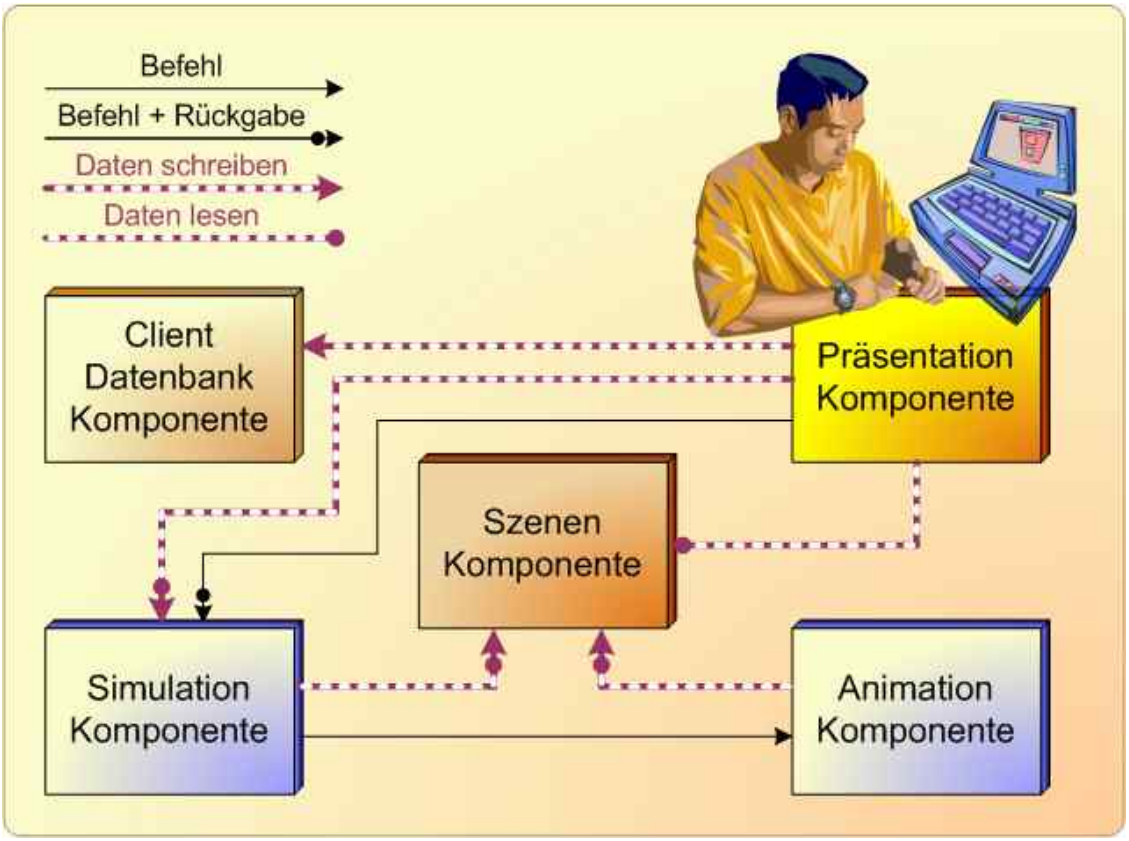

Abbildung 6.4: *Die Präsentation-Komponente visualisiert die Szene und gibt mögliche Benutzereingaben an die Simulation-Komponente des Servers weiter. Die Simulation-Komponente berechnet die Simulation und sendet der Animation-Komponente daraus resultierende Befehle zur Ausführung von Animationen.*

log zur Multiplexer-Komponente kommuniziert auch die Demultiplexer-Komponente mit der Codec-Manager-Komponente für die Bereitstellung passender Codecs, über deren Hilfe sie die Datenströme dekodiert. Sobald ein empfangenes Element sich in einem Zustand befindet, der seine Visualisierung erlaubt, trägt die Demultiplexer-Komponente das Element in die Szenen-Komponente ein. Vollständig rekonstruierte Animationsvorschriften werden ebenfalls in der Szenen-Komponente des Clients vermerkt.

Gegenüber Abbildung 6.2 fällt hier eine eindeutige Definition von Daten und Kommandos schon schwerer. Ein Beispiel ist die Übergabe der selektierten Elemente von der Scheduler-Komponente zur Multiplexer-Komponente. Der ausschlaggebende Grund für eine Kommandoverbindung ist letztlich die geringe Menge an Informationen, die pro Element anfällt.

Nach dem Laden und Übertragen der Szene ist die Visualisierung der Szene auf dem Client der nächste Schwerpunkt. Wie in Abbildung 6.4 zu sehen, stellt dabei die Präsentation-Komponente einen entscheidenden Faktor dar: Sie liest die Daten aus der Szenen-Komponente und zeigt diese auf dem Endgerät des Benutzers an. Weiterhin bietet sie im Rahmen einer geräteübergreifenden Kommunikation mit der Simulation-Komponente die Möglichkeit zur Beeinflussung der Simulation, welche die dynamischen Vorgänge der Szene berechnet. Die Entscheidung zwischen einer Daten- beziehungsweise Kommandoverbindung ist vielleicht an dieser Stelle am schwierigsten, da die Präsentation-Komponente einerseits lediglich ei-

ne Matrix mit der Aufforderung zur Transformation eines Elements, andererseits aber auch komplexere simulationsspezifische Informationen übermitteln kann. In Abbildung 6.4 sind deshalb beide Kanäle verzeichnet. Zusätzlich ist ebenfalls ein lesender Zugriff von Seiten der Präsentation-Komponente erlaubt, um beispielsweise den aktuellen Status der Simulation zu erfragen. Weil neben der Interaktion mit dem Benutzer eine weitere Aufgabe der Präsentation-Komponente in der Fütterung der Client-Datenbank-Komponente mit Daten besteht, ist zwischen den Komponenten eine geräteübergreifende Datenverbindung eingetragen.

Während der Simulation liest die Simulation-Komponente den aktuellen Status der Szenen-Komponente und berechnet darauf basierend den nächsten Zeitschritt der Simulation. Eventuelle Änderungen registriert sie wieder in der Szenen-Komponente des Servers. Weiterhin wertet sie für die betroffenen Elemente einfache Animationsbefehle aus und sendet diese in einer ebenfalls geräteübergreifenden Kommandoverbindung an die Animation-Komponente des Clients. Die Animation-Komponente verwendet die in der Szenen-Komponente eingetragenen Animationsvorschriften, um die Elemente der Szene zu transformieren, d.h. es wird sowohl eine lesende als auch eine schreibende Datenverbindung von Seiten der Animation-Komponente zur Szenen-Komponente benötigt.

Während in den beiden letzten Abschnitte die Aufgaben der Komponenten als auch ihre Kommunikation untereinander beschrieben wurden, stellen die folgenden Abschnitte nun die Komponenten selbst vor. Dabei wird unter anderem analysiert, welche Anforderungen aus Abschnitt 5.1 von der jeweiligen Komponente betroffen sind. Existiert eine Komponente auf Client und Server so wird in diesen Abschnitten auf die jeweiligen Unterschiede eingegangen.

6.3 Die Szenen-Komponente

Die Aufgabe der Szenen-Komponente ist die Repräsentation einer großen und dynamischen 3D Szene. Sie kapselt nicht nur die visuellen und verhaltensspezifischen Informationen, sondern auch die räumliche Sortierung der Elemente.

6.3.1 Der Animationsagent

Kern der Repräsentation ist der in Abschnitt 5.1.1 eingeführte Animationsagent, welcher nach dem Prinzip der Animationselemente sowohl das äußere Erscheinungsbild als auch das Verhalten eines Elements der Szene beinhaltet. Grundsätzlich unterscheidet der Server zwischen aktiven, passiven und statischen Animationsagenten. Erstere sind von sich aus in der Lage, vordefinierte Aktionen auszuführen und auf eintretende Ereignisse mit bestimmten Folgehandlungen zu reagieren. Passive Animationsagenten verweilen normalerweise in einem statischen Zustand, können aber auf Antrieb eines aktiven Agenten vorübergehend in eine zeitlich begrenzte Phase eintreten, während der sie aktiven Animationsagenten entsprechen. Statische Animationsagenten hingegen stellen ein gewisses Paradoxon zur üblichen Definition autonomer Agenten [BS01] dar, indem sie während der gesamten Zeitdauer der Simulation keine Transformation ihres Zustands erlauben. Im Falle eines Fußballspiels etwa wären die

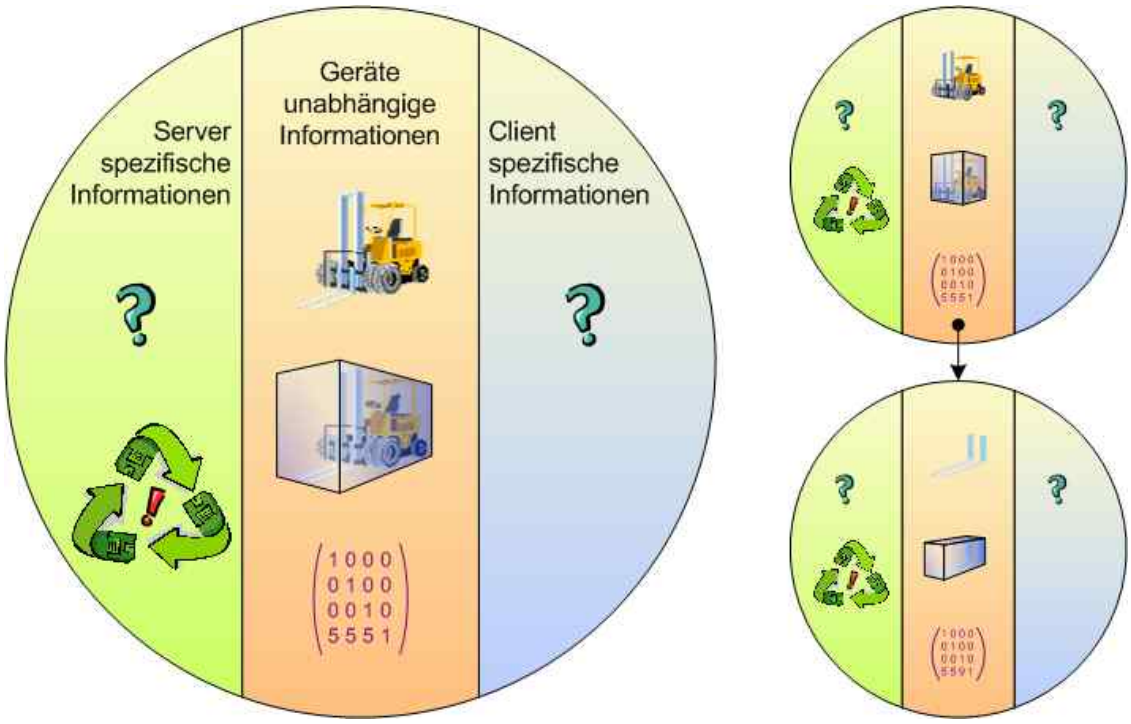

Abbildung 6.5: *Der Animationsagent ist der zentrale Baustein einer dynamischen Szene. Er kapselt visuelle sowie verhaltenspezifische Informationen. Je nach seiner aktuellen Lage verfügt der Animationsagent über client- beziehungsweise serverspezifische Informationen. Beispielsweise haben die Agenten in der Regel auf Server und Client unterschiedliche Identifikationen, da die Clients lediglich eine Teilmenge der Szene verwalten.*

Spieler aktive Animationsagenten, der Ball ein passiver Animationsagent und die Tore statische Animationsagenten. Im Unterschied zum Server kennt der Client nur dynamische und statische Animationsagenten. Die Ursache hierfür liegt darin begründet, dass der Client keine Simulation zu berechnen hat. Stattdessen wendet er lediglich Animationsvorschriften auf die aus seiner Sicht dynamischen Agenten an. Die Interpretation statischer Animationsagenten ist auf beiden Seiten identisch.

Die unterschiedliche Auslegung der Animationsagenten auf Server und Client wirkt sich bis auf das in Abbildung 6.5 dargestellte Konzept der Agenten aus. Hier ist eine Dreiteilung in client- beziehungsweise serverspezifische Informationen sowie in geräteunabhängige Informationen zu erkennen. Wohingegen auf Seiten des Servers noch die durch die drei grünen Pfeile symbolisierte Verhaltensbeschreibung existiert, fehlt das entsprechende Äquivalent auf Seiten des Clients. Hierzu sei an die in Abschnitt 6.2 beschriebene Rollenverteilung zwischen Simulation-Komponente und Animation-Komponente erinnert. Erstere muss für die Berechnung der Simulation genaue Kenntnis darüber haben, welches Verhalten welchem Agenten zugeordnet ist. Sobald sie eine Aktion eines Animationsagenten detektiert hat, sendet sie der Animation-Komponente sowohl eine eindeutige Identifikation des betroffenen Agenten als auch der auszuführenden Aktion. Die Animation-Komponente selektiert daraufhin anhand der beiden Identifikationen Agent und Aktion aus der Szenen-Komponente, d.h. die Zuordnung von Verhalten und Animationsagent ist auf Seite des Clients nicht von vornherein festgelegt, sondern erfolgt dynamisch zur Laufzeit. Die eindeutige Identifikation eines

Animationsagenten ist in Abbildung 6.5 durch die Fragezeichen symbolisiert und kann auf Server und Client unterschiedlich ausfallen. Der Grund ist in der oftmals kleinen Teilmenge zu suchen, welche ein Client aufgrund seiner im Vergleich zum Server geringeren Leistungsmerkmale von einer Szene aufrecht erhält.

In den Bereich der unabhängigen Informationen eines Animationsagenten fällt die Beschreibung des äußeren Erscheinungsbildes. In Abbildung 6.5 handelt es sich dabei um einen Gabelstapler. Wohingegen der Server die Informationen für die Analyse der an einen Client zu übertragenden Datenströme benötigt, macht der Client im Rahmen der Visualisierung von den empfangenen Daten Gebrauch. Eine von dem Erscheinungsbild des Animationsagenten abhängige Information ist die Bounding Volume, welche eine quaderförmige, achsenparallele, konvexe Hülle um die Geometrie des Animationsagenten beschreibt. Im folgenden werden derartige Bounding Volumes gemäß dem englischem Begriff der *Axis Aligned Bounding Box* abgekürzt *AABB* genannt. Andere Formen der Volumen wie etwa Zylinder, Kugeln oder *Oriented Bounding Boxes* bieten zwar in bestimmten Fällen eine bessere Approximation an das Modell, implizieren aber in vielen Anwendungen wie Occlusion Culling oder Kollisionserkennungen auch kostspieligere Tests. Während das durch den Animationsagenten repräsentierte Modell in Objektkoordinaten definiert ist, beinhaltet die AABB Weltkoordinaten. Eine Transformation des Modells in Weltkoordinaten erfolgt mit Hilfe der ebenfalls im Animationsagenten eingetragenen Transformationsmatrix.

Animationsagenten können nach dem Prinzip der Animationshierarchien zu einem Baum strukturiert werden. Die Kindagenten erben dabei durch Aufmultiplizieren der Matrizen die Transformationen ihrer Vorgänger in der Hierarchie. Abbildung 6.5 zeigt auf der rechten Seite eine weitere Variation des Gabelstaplers. Hierbei wird der fahrbare Untersatz von der Wurzel der Animationshierarchie repräsentiert und die ebenfalls bewegliche Gabel von einem Kindagenten. Bewegt sich der Untersatz, so folgt die Gabel der Bewegung. In der Regel beinhaltet die Wurzel einer Animationshierarchie das eigentliche Verhalten des Gesamtmodells, wohingegen die Kinder lediglich einfache Aktionen ausführen können.

6.3.2 Die visuelle Repräsentation

Jeder Animationsagent verweist auf einen sogenannten *Elementgraphen*, welcher über eine azyklische Hierarchie das äußere Erscheinungsbild des Agenten definiert. Das Prinzip ist analog zu dem in Abschnitt 4.7 erläuterten Zustandsmodell der Szenegraphen. Im Unterschied zu den dortigen Graphen wie etwa VRML oder Performer bildet der Elementgraph allerdings ein statisches Gebilde, d.h. er verfügt weder über Sensoren noch über andere Knoten, die über die Beschreibung optischer Informationen hinausgehen. Die Knotentypen eines Elementgraphen lassen sich in die folgenden Gruppen gliedern:

- **Gruppenknoten**: Gruppenknoten sind die einzigen Knoten des Elementgraphen, die andere Knoten als Kinder referenzieren können. Ähnlich dem Konzept anderer Szenegraphen restaurieren sie den ursprünglichen Zustand nach der Traversierung eines Teilbaumes. Typischerweise ist die Wurzel eines Elementgraphen ein Gruppenknoten. Als besonderer Knoten beinhaltet die Wurzel noch eine AABB der gesamten Geometrie des Graphen.

- **Transformknoten**: Jeder Transformknoten beinhaltet eine in der Regel konstante Matrix, welche die Transformation der in den nachfolgenden Knoten beschriebenen Geometrie ermöglicht. Veränderungen dieser Art sind ein typisches Beispiel für die Zustandsrestaurierung von Seiten der Gruppenknoten, da die Transformation durch den übergeordneten Gruppenknoten wieder rückgängig gemacht wird. Die Verwendung der Transformknoten ist nicht zu verwechseln mit dem Erstellen von Animationshierarchien, für das sich die Animationsagenten selbst verantwortlich zeichnen. Die Transformknoten dienen lediglich der Modellierung eines Gebildes über die Kombination mehrerer Basiskörper. So ist es beispielsweise möglich, ein Modell nur mit Hilfe transformierter Würfel zu beschreiben.

- **Geometrieknoten**: Geometrieknoten repräsentieren eine Menge von Koordinaten. Dabei kann es sich um Scheitelpunkte, Normalen oder Texturkoordinaten handeln. Die Dimension ist somit nicht zwangsläufig vorgeschrieben.

- **Topologieknoten**: Topologieknoten beschreiben die Konnektivität der in den Geometrieknoten vermerkten Scheitelpunkte eines Modells. Insofern muss ein auf einen Geometrieknoten folgender Topologieknoten die entsprechende Anzahl an Scheitelpunkten referenzieren.

- **Farbenknoten**: Farbenknoten beschreiben die Farben eines Modells.

- **Materialknoten**: Materialknoten beinhalten besondere Eigenschaften eines Modells, welche unter anderem für Lichtberechnungen von Bedeutung sind. Typische Merkmale sind etwa die Reflexion oder die Transparenz des Modells.

- **Texturknoten**: Jeder Texturknoten repräsentiert exakt eine Textur, welche auf das Modell gelegt wird.

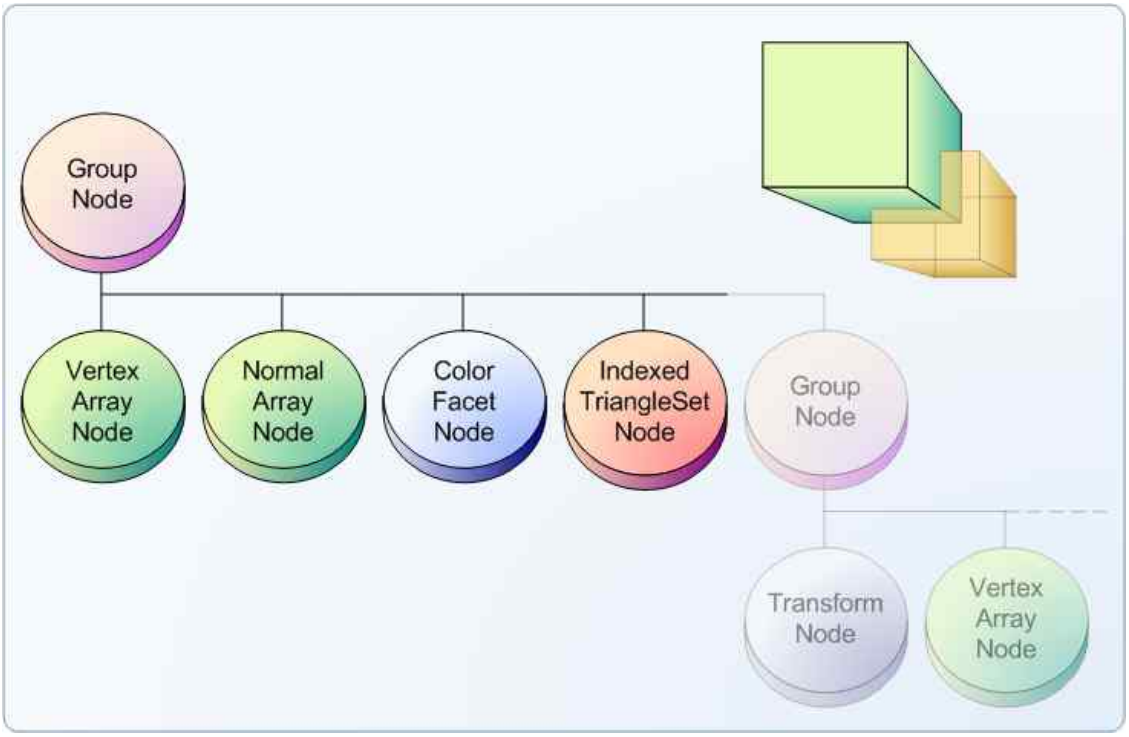

Abbildung 6.6: *Der Elementgraph dient der flexiblen Beschreibung der visuellen Informationen eines Animationsagenten. Während einer Simulation bleibt ein Elementgraph normalerweise unverändert, weil die Animationsagenten die dynamischen Bausteine einer Szene kapseln.*

Abbildung 6.6 illustriert ein Beispiel eines Elementgraphen. Eine zusätzliche Eigenschaft der dort aufgeführten Knoten ist die Option, Daten sowohl pro Scheitelpunkt als auch pro Fläche spezifizieren zu können. Entsprechende Änderungen haben weitreichende Konsequenzen auf das äußere Erscheinungsbild eines Modells. In dem Beispiel repräsentiert der erste Teil des Elementgraphen den grünen Würfel, während der nach dem Gruppenknoten angedeutete Teilbaum den gelben Würfel beschreibt.

Da in vielen Szenen Redundanzen auftreten, wäre eine direkte Speicherung der visuellen Informationen innerhalb eines Elementgraphen hinsichtlich des Speicherverbrauchs nicht sonderlich effizient. Warum soll der bereits als Beispiel angeführte Stanford Bunny mit allen Informationen zweimal gespeichert werden, obgleich sich beide Versionen lediglich in der Farbe unterscheiden? Anwendungen, in denen typischerweise eine große Zahl derartiger Redundanzen auftreten, sind Flugsimulatoren, die aufgrund der riesigen abzudeckenden Welt für die Gebäudemodellierung oftmals stereotypische Grundkörper einsetzen. Aus diesem Grund erfolgt die Konservierung der Daten, wie in Abbildung 6.7 dargestellt, in sogenannten *Pools*. Jeder Pooleintrag beschreibt genau eine bestimmte Eigenschaft, also etwa eine Textur, eine Topologie oder eine Menge von Scheitelpunkten. Die Knoten eines Elementgraphen referenzieren lediglich die Pooleinträge. Analog zu den Knotentypen gibt es eine Anzahl verschiedener Pooltypen: Der *Geometry Pool* beinhaltet geometrische Informationen wie etwa Normalen, Scheitelpunkte oder Texturkoordinaten, der *Topology Pool* topologische Informationen, der *Color Pool* Farbinformationen, der *Material Pool* Materialeigenschaften und der *Texture Pool* Texturen. Mit der Ausnahme von Transformknoten und Gruppenknoten

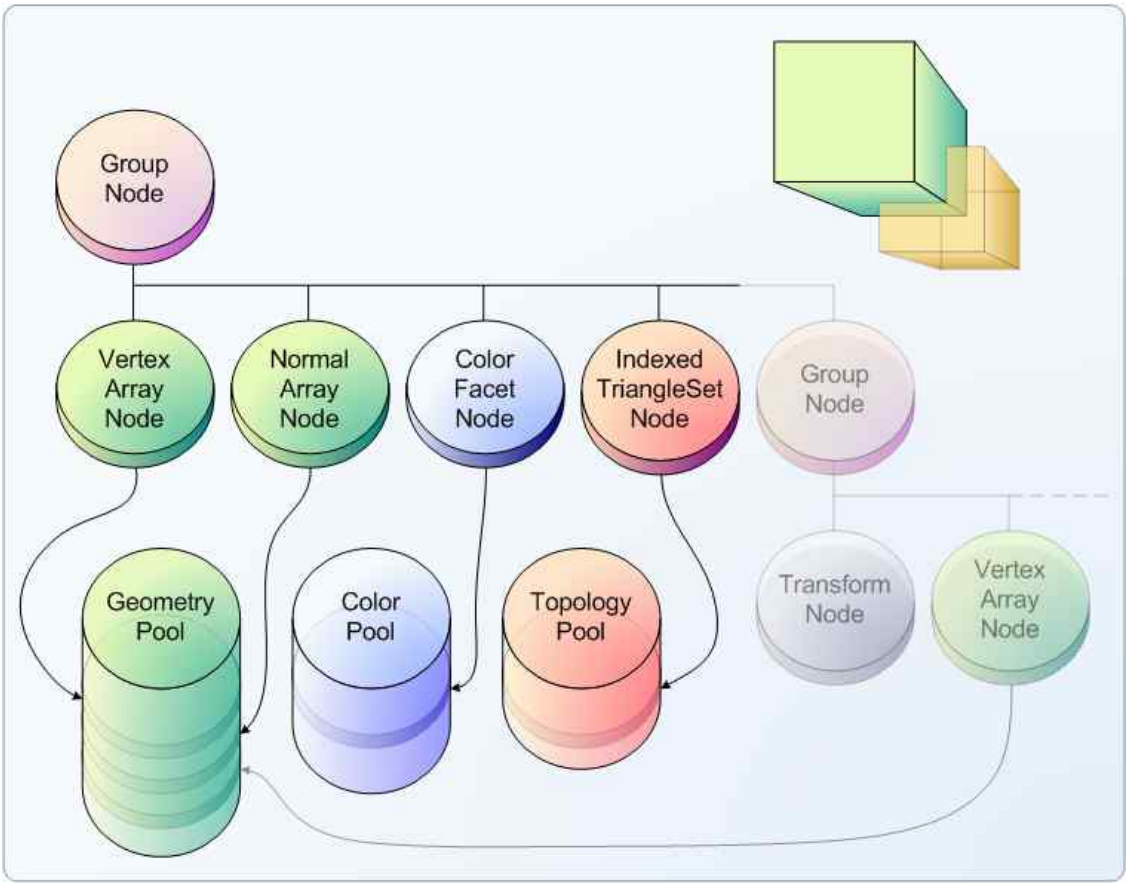

Abbildung 6.7: *Aus Effizienzgründen speichern die Elementgraphen nicht direkt die visuellen Informationen, sondern referenzieren entsprechende Pooleinträge. Mehrere Elementgraphen können sich die gleichen Pooleinträge teilen, genauso wie ein Elementgraph denselben Pooleintrag mehrmals adressieren darf.*

können somit alle Knoten eines Elementgraphen entsprechende Einträge referenzieren. Die Einführung eines Pool für Matrizen macht aufgrund des schlechten Verhältnisses zwischen anfallender Speichermenge pro Matrix und möglichen Redundanzen wenig Sinn. Anders sieht es dagegen im Falle der Gruppenknoten aus. Hier ist zumindest bezüglich der Wurzelknoten ein weiterer Pool erforderlich, mit dessen Hilfe ganze Modelle effizient referenziert werden können. Beinhalten mehrere Wurzelknoten identische Elementgraphen, so wird der entsprechende Graph nur einmal innerhalb des *Element Graph Pools* abgelegt.

Eine positive Eigenschaft der Pool ist die Tatsache, dass sie abgesehen von den Informationen der Animationsagenten sämtliche visuellen Informationen einer Szene speichereffizient kapseln. Wenn es möglich ist, die Einträge der Pools an die Clients zu übertragen, dann ist ein großer Teil der Arbeit bereits getan. Ziel muss dabei sein, die speichereffiziente Repräsentation auch wieder auf dem jeweiligen Client zu rekonstruieren, selbst wenn die einzelnen Einträge selektiv oder in beliebiger Reihenfolge transferiert werden. Abbildung 6.8 gibt einen Überblick auf die Struktur der Repräsentation einer Szene: Jeder Animationsagent referenziert für seine visuelle Beschreibung einen Eintrag innerhalb des Element Graph Pools. Der entsprechende Eintrag verweist auf die Wurzel eines Elementgraphen, dessen Knoten wiederum Einträge in den restlichen Pool indizieren.

Ähnlich den Animationsagenten gibt es auch bei den Pooleinträgen Unterschiede zwischen Server und Client, da der Server für jeden Pooleintrag darüber Buch führen muss, auf welchen Clients der betreffende Eintrag momentan lokal zur Verfügung steht. Eine entsprechende Verwaltung ist auf Seiten des Clients nicht nötig. Die visuelle Beschreibung der Modelle an sich, d.h. sowohl die Elementgraphen als auch die zugehörigen Informationen, ist von diesen Differenzen nicht betroffen.

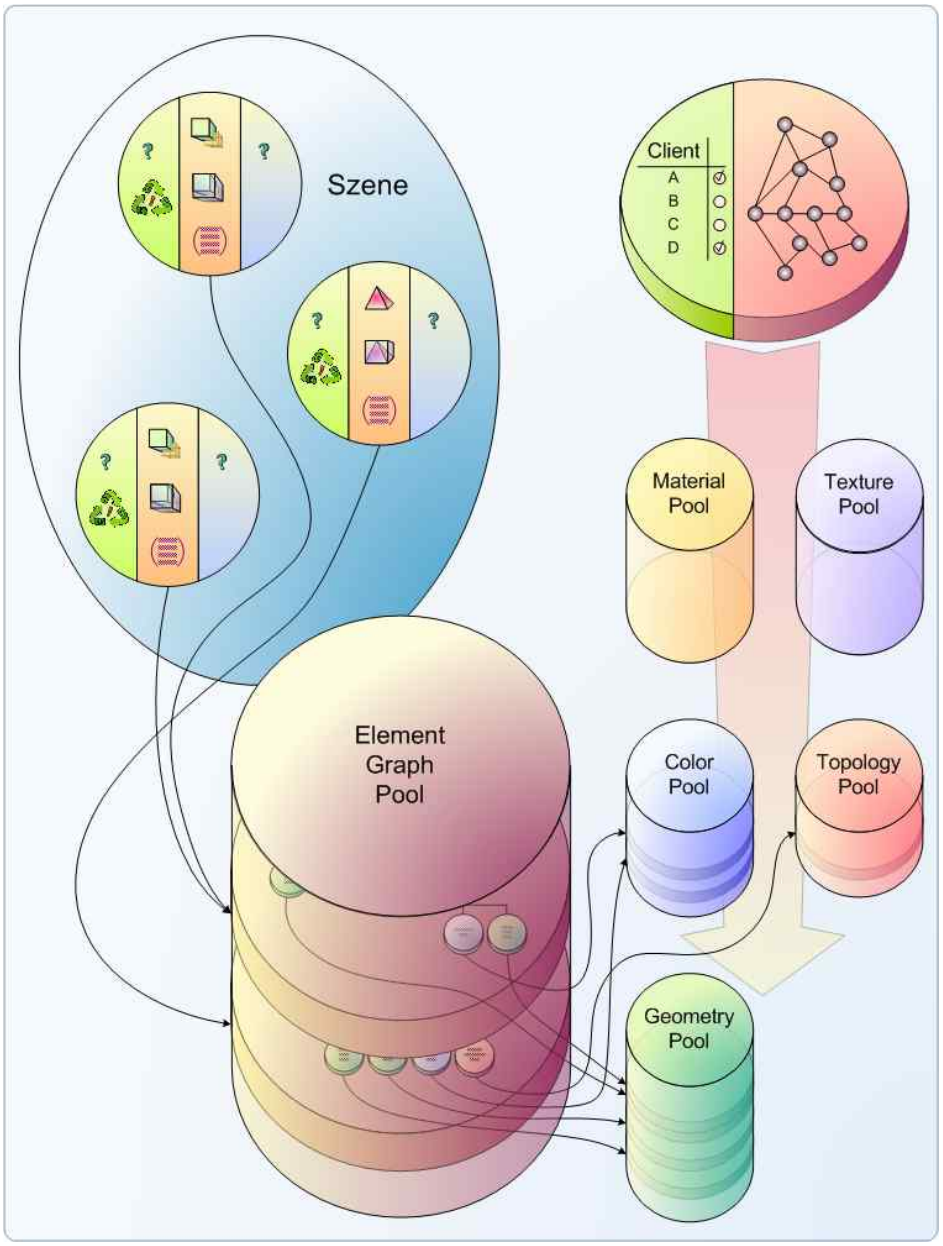

Abbildung 6.8: *Ein Überblick auf die Repräsentation einer Szene ohne deren räumliche Sortierung. Die Animationsagenten referenzieren Elementgraphen, die wiederum Pooleinträge adressieren. Da die Agenten sich dieselben Elementgraphen teilen können, gibt es für letztere ebenfalls einen Pool. Pooleinträge verfügen ähnlich wie Animationsagenten über client- und serverspezifische Informationen. Beispielsweise registriert der Server, an welche Clients ein Pooleintrag bereits übermittelt wurde.*

6.3.3 Die verhaltensspezifische Repräsentation

Dynamische, interaktive Szenen erlauben in der Regel lediglich einen begrenzten Freiheitsgrad hinsichtlich dynamischer Abläufe, weil sämtliche Vorgänge einer Animation oder Simulation bereits in einem Zustandsmodell vordefiniert sind. Die Übergänge zwischen Animation und Simulation sind häufig nicht eindeutig abzugrenzen, weshalb der folgende Text davon ausgeht, dass der allgemeinere Begriff der Simulation mögliche Animationen mit einschließt.

Zu Beginn einer Simulation am Startzeitpunkt t_0 befindet sich jeder Animationsagent in Bezug auf das Zustandsmodell in einem Ausgangszustand $z_0(t_0)$. Wohingegen statische Animationsagenten diesen Zustand während der gesamten Dauer der Simulation nicht verlassen, steht aktiven und passiven Animationsagenten auf Seiten des Servers jeweils eine Menge von Zuständen $Z = \{z_0, \ldots, z_n\}$ zur Verfügung, die sie im Rahmen der Simulation einnehmen können. Zusätzlich definiert die Simulation eine Abbildung der Menge Z auf Z selbst, d.h. für jeden Zustand z_x mit $x = 0, \ldots, n$ gibt es eine möglicherweise leere Menge von Operationen $R = \{r_1, \ldots, r_k\}$ für die gilt $r_i(z_x) = z_y$ mit $i = 1, \ldots, k$ und $z_y \in Z$. Die Frage ist nun, wann welche Operation auf den aktuellen Zustand eines Animationsagenten angesetzt wird. In einer nicht interaktiven Simulation [2] kann diese Frage einfach mit einer Funktion $f(t)$ über die Zeit t beantwortet werden, d.h. während der gesamten Dauer der Simulation von t_0 bis t_{end} gilt $r = f(t)$ mit $t = t_0, \ldots, t_{end}$ und $r \in R$. Schwieriger wird es jedoch, wenn im Falle einer interaktiven Simulation der Benutzer den Ablauf zu beliebiger Zeit beeinflussen darf. In diesem Zusammenhang muss das Eingreifen des Benutzers als Teil einer Situation aufgefasst werden, die als Reaktion die Anwendung einer oder mehrerer Operationen impliziert. Die Funktion f ist also nicht über die Zeit, sondern über die spezifizierten Situationen definiert. Eine Situation S entspricht dabei einer Menge von Zuständen, die zu einem Zeitpunkt t alle gegeben sein müssen. Die Elemente aus S können unter anderem aus der Vereinigung

$$\bigcup_{j=1}^{h} Z_j \tag{6.1}$$

aller Zustandsmengen Z_j rekrutieren mit $j = 1, \ldots, h$ und h gleich der Anzahl Animationsagenten. Es existieren aber eventuell noch weitere simulationsspezifische Zustände, die unabhängig vom Status der Animationsagenten sind.

Auf einer weniger formalen Ebene beinhaltet die Menge R die potentiellen Aktionen beziehungsweise Reaktionen eines Animationsagenten auf eine bestimmte Situation S. Das Problem liegt dabei oftmals in der Identifikation einer Situation als solche, da eine Überprüfung der Zustände bezüglich des Rechenaufwands äußerst kostspielig werden kann. Grundsätzlich ist sowohl das Testen der Zustände als auch die Bereitstellung der entsprechenden Algorithmen Aufgabe der Simulation. Nach Abschnitt 5.1.3 soll aber zumindest eine Schnittstelle zur Beschreibung dynamischer Vorgänge bereitgestellt werden, wozu Abbildung 6.9 einen ersten Überblick verschafft. Ähnlich dem Ansatz anderer Szenegraphen wie VRML oder DIVE erfolgt die Beschreibung des Verhaltens mit Hilfe von Sensoren. Sensoren sind Komponenten,

[2] In diesem Fall wäre der Begriff Animation vielleicht eher angebracht.

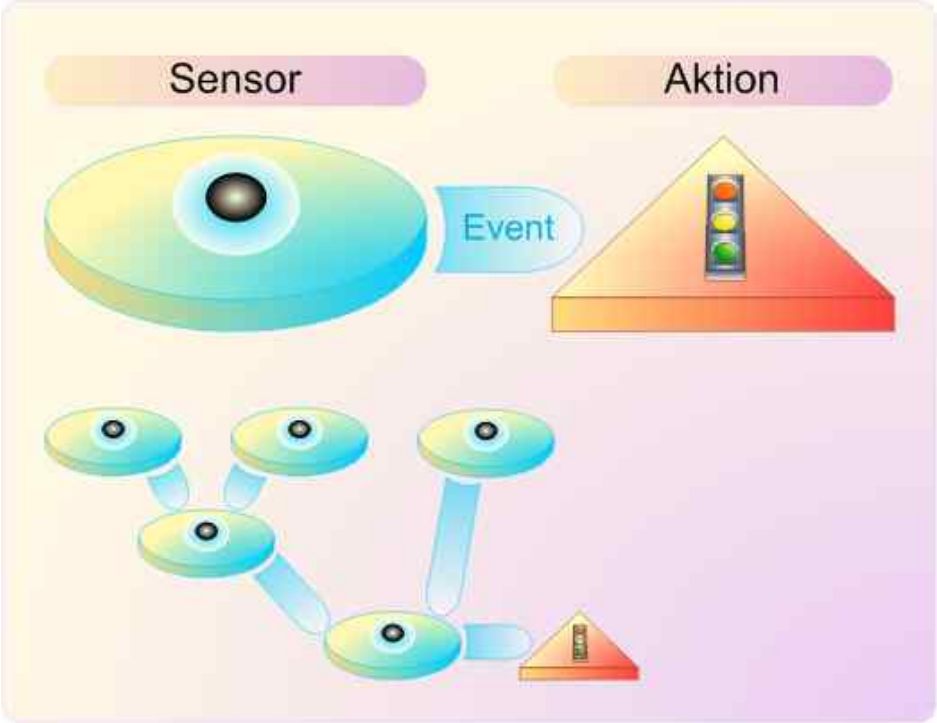

Abbildung 6.9: *Ähnlich zu VRML wird das Verhalten durch Ereignisquellen und Ereignissenken geprägt. Da Sensoren als Ereignisquellen gleichzeitig Ereignissenken darstellen können, ist eine Hierarchie von Sensoren und somit eine Beschreibung von komplexen Ereignissen möglich.*

die in der Lage sind, eine bestimmte Situation zu erkennen und eine entsprechende Information in Form eines Ereignisses oder eines *Events* an andere Interessenten weiterzuleiten. Bei diesen Interessenten handelt es sich um Komponenten, welche die empfangenen Ereignisse auswerten und gegebenenfalls in Reaktionen umsetzen können. Sie werden im folgenden auch als Ereignissenken bezeichnet. Gleichwohl Sensoren zu den Ereignisquellen zu zählen sind, können sie ebenfalls Ereignissenken darstellen, wodurch wie in Abbildung 6.9 illustriert eine Graphenstruktur der Sensoren möglich ist. Sensoren erlauben somit gleich einem Baukastenprinzip das Detektieren immer komplexerer Situationen. Beispielsweise ist durch die Verbindung zweier Sensoren eine logische *Und* Verknüpfung denkbar.

Hinsichtlich der Animationsagenten repräsentieren die Ereignissenken üblicherweise diejenigen Instanzen, welche in Abhängigkeit von einer Situation S die geeignete Operation $r \in R$ auswählen. Das Verhalten eines Animationsagenten ist dadurch als ein Graph von Ereignisquellen und Ereignissenken definiert. Derartige Graphen existieren nur auf Seite des Servers und sind in der Simulation-Komponente gekapselt. Für die Szenen-Komponente ist lediglich eine Unterart der Operationen von Interesse, nämlich die Animationen. Hierbei handelt es sich um vordefinierte Transformationsbeschreibungen, die als Reaktion auf ein Ereignis ausgeführt werden. Der Status einer Animation ist über die *Animationszeit* spezifiziert und aus diesem Grund abgesehen vom Startzeitpunkt der Animation unabhängig vom aktuellen Zeitpunkt der Simulation. Die Animationszeit beginnt mit dem Start der Animation und endet nach einer für die betreffende Animation fest vorgegebenen Zeitdauer. Abbildung 6.10 gibt einen Überblick auf das Konzept einer Animation [EaSL03]. Grundlegender

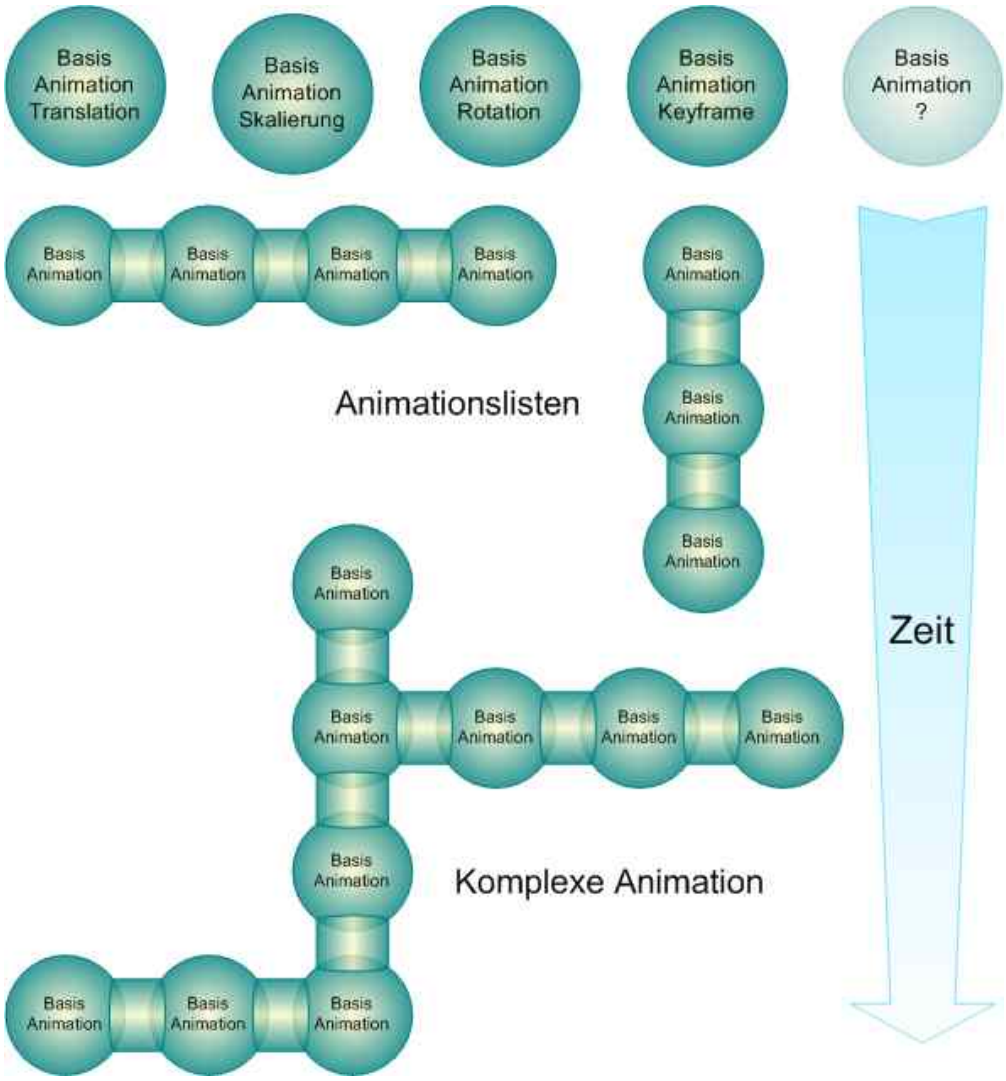

Abbildung 6.10: *Grundlegende Bausteine einer Animation sind die Basisanimationen, welche zu Animationslisten kombiniert werden können. Animationslisten bieten die sequentielle als auch die parallele Abarbeitung ihrer Inhalte und ermöglichen die Verknüpfung zu komplexen Animationen bis hin zu einer aufgabenorientierten Ebene.*

Baustein sind sogenannte *Basisanimationen* wie etwa Translation, Skalierung und Rotation. Eine weitere Basisanimation stellen Keyframes dar, die ähnlich einem Film abgespielt werden. Basisanimationen erlauben die Kombination zu *Animationslisten*. Eine Besonderheit der Animationslisten liegt in deren Option zur parallelen oder sequentiellen Abarbeitung. Hierdurch ist die gleichzeitige Anwendung mehrerer Basisanimationen möglich. Da Animationslisten selbst wiederum Animationen repräsentieren, können sie zu *komplexen Animationen* ineinander verschachtelt werden. Dieses Konzept offeriert die Abstraktion der Animationen auf ein benutzerfreundliches Level. Angenommen ein Benutzer möchte den Greifarm eines Roboters modellieren, wie es Abbildung 6.11 darstellt. Hierzu benötigt er eine Animationshierarchie aus drei Animationsagenten. Die Bewegungen der einzelnen Glieder sind in diesem Beispiel auf einfache Rotationen beschränkt und können in einer parallelen Animationsliste kombiniert werden. Jede Animation identifiziert also nicht nur einen bestimmten Bewegungs-

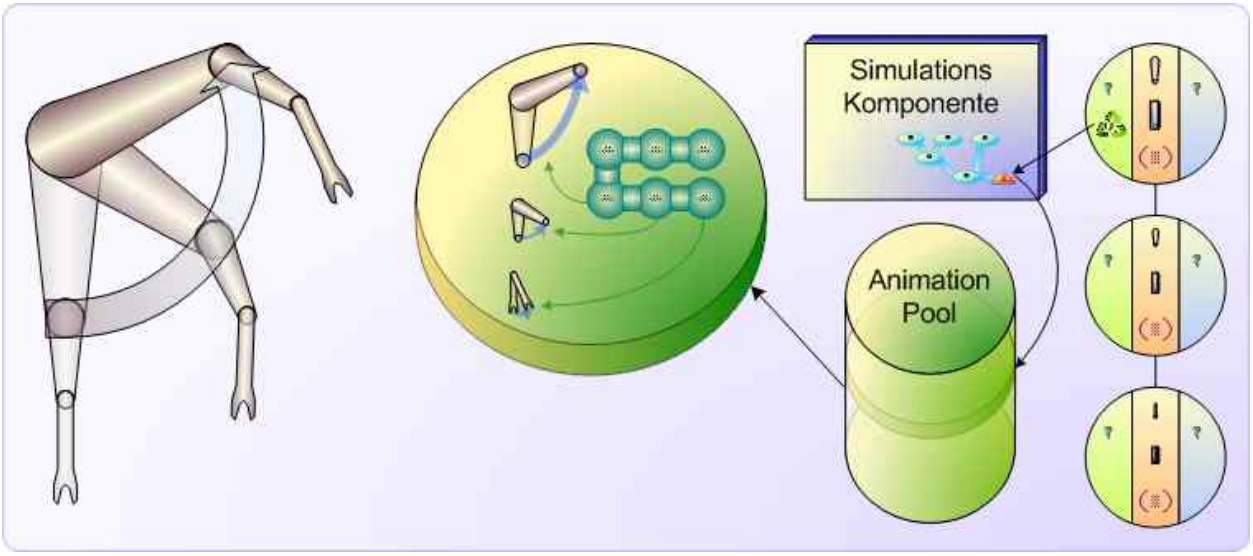

Abbildung 6.11: *Ähnlich den visuellen Informationen werden Animationsvorschriften in einem Pool abgelegt. Sie entsprechen den Informationen einer Simulation, die im Laufe einer Session eventuell an einen Client übertragen werden.*

ablauf, sondern auch den korrespondierenden Knoten innerhalb der Animationshierarchie. Als Konsequenz verweist nur die Wurzel der Animationshierarchie auf eine Animation. Der Greifarm soll nun einen Gegenstand aufheben und an einer anderen Position absetzen. Der Benutzer definiert daher zwei parallele Animationslisten mit jeweils drei Basisanimationen. Jede dieser Basisanimationen entspricht der Rotation eines Gliedes des Greifarms. Um der Animationshierarchie Rechenschaft zu leisten, sind die Animationen mit aufsteigendem Level des betroffenen Agenten von links nach rechts angeordnet. Anschließend kombiniert der Benutzer die beiden parallelen Animationslisten mit Hilfe einer sequentiellen Animationsliste zu einer komplexen Animation, welche dem Umsetzen eines Gegenstandes durch den Roboter entspricht. Der Anwender muss sich nun nicht mehr mit den Details der Animation befassen, da der Roboter aus seiner Sicht lediglich eine Operation *Umsetzen* anbietet. Ähnlich dem Prinzip der Objektorientierung ist es sogar möglich, die Animation des Roboters auf andere Animationshierarchien mit gleicher Topologie zu übertragen. Hierzu müssen nur die Basisanimationen entsprechend angepasst werden.

Analog zu den visuellen Daten sind auch hinsichtlich der Animationen Redundanzen möglich. Es wäre beispielsweise unnötig, für zwei identische Animationsagenten beide Male die gleiche Animation zu definieren. Aus diesem Grund beinhaltet die Szenen-Komponenten einen weiteren Pool, nämlich den *Animation Pool*. Dieser beinhaltet alle Animationen, die während einer Simulation auftreten können. Gleichzeitig repräsentiert der Animation Pool exakt die Daten, die für die Animierung der Szene auf Seite des Clients benötigt werden. Folglich existiert der Animation Pool sowohl auf Seite des Servers als auch auf Seite des Clients. Sobald der Server einen Agenten für die Übertragung selektiert, überprüft er die zu dem Agenten korrespondierenden Animationen. Sind diese auf dem Client noch nicht vorhanden, so müssen sie ebenfalls transferiert werden.

Für die Identifizierung der zu einem Animatiomsagenten gehörenden Animationen muss der

Server den Umweg über die Simulation-Komponente gehen. Auf Seite des Servers referenziert jeder Animationsagent einen Graph aus Ereignisquellen und -senken, wobei letztere die Einträge des Animation Pool referenzieren. Anhand der Simulation erkennt der Server die momentan benötigte Animation und kann darüber eine Priorisierung der zu übertragenden Animationen ermitteln. Wenn es gelingt, die Agenten gemäß einem Area-of-Interest Konzept zu selektieren, dann ist auf diese Weise die in Abschnitt 5.1.3 geforderte skalierbare Übertragung der dynamischen Vorgänge gewährleistet. Auf Seite des Clients verweisen die Agenten nicht auf Einträge des Animation Pool. Die Zuordnung zwischen Animation und Agent erfolgt hier über die Kommunikation von Simulation-Komponente und Animation-Komponente.

6.3.4 Die räumliche Sortierung

Ein auffälliges Merkmal an Abbildung 6.8 ist die fehlende räumliche Zuordnung der Animationsagenten innerhalb der Szene. Entsprechende Techniken wurden bereits in Form der Bounding Volume Hierarchies und der Raumunterteilungbäume im Kapitel 4 vorgestellt. Abschnitt 5.1.1 diskutiert die Vor- und Nachteile beider Ansätze und erläutert, warum zumindest innerhalb dieser Arbeit einer Variante der Raumunterteilungsbäume der Vorzug zu geben ist. Der Einsatz der Raumunterteilungsbäume impliziert hinsichtlich großer, dynamischer Szenen zwei essentielle zu lösende Probleme: Zum einen die effiziente Verwaltung von Schnittelementen und zum anderen die konsistente Einhaltung der Kapazität δ jeder Zelle innerhalb des Baumes. Ein Hintergedanke der Kapazität δ liegt unter anderem darin, eine maximale Beschränkung für die lineare Auflösung einer Problemstellung vorzugeben. Als Beispiel kann hier wieder die Suche nach einem Element der Szene unter Vorgabe einer Position dienen. Die schnelle Identifizierung der zur Position korrespondierenden Zelle ist aufgrund der Auswertung der durch einen Raumunterteilungsbaum repräsentierten räumlichen Kohärenzen gegeben. Innerhalb der Zelle müssen die Elemente jedoch in einem linearen Suchvorgang überprüft werden, wobei δ die Anzahl der Elemente und somit den linearen Aufwand der Suche definiert. Üblicherweise benötigt eine Applikation für das Lösen derartiger Probleme zwei grundlegende Arten von Tests: Einerseits in Bezug auf eine Zelle des Raumunterteilungsbaum und andererseits in Bezug auf die in den Baum einsortierten Elemente. Häufig ist letzterer Test der genauere und damit auch aufwändigere. So ist etwa im Falle des obigen Suchbeispiels die Überprüfung, ob sich die Position innerhalb einer Zelle befindet, äußerst einfach. Für einen vergleichbaren exakten Test der Elemente muss dagegen deren gesamte Geometrie analysiert werden. Während nun kleinere Kapazitäten eine größere Tiefe des Baumes und somit mehr Zelltests bewirken, implizieren größere Werte eine flachere Hierarchie und folglich aufgrund des höheren linearen Anteils eine größere Zahl der Elementtests. Insofern ist für jede Applikation eine Abschätzung erforderlich, in welcher Relation der Aufwand eines Zelltests zu einem Elementtest steht. Das Resultat kann von Applikation zu Applikation unterschiedlich ausfallen, da beispielsweise für Sichtbarkeitstest im Rahmen eines Occlusion Cullings eine entsprechende Hardwareunterstützung vorhanden ist. Eine derartige Hilfeleistung erhalten Kollisionserkennungen leider nicht. In Fällen wie etwa besagtem Suchvorgang spielt auch die Komplexität der Geometrie der Elemente eine entscheidende Rolle. Die Frage nach dem optimalen Wert für δ muss also leider negativ be-

schieden werden, da die gewählte Kapazität sowohl von den Eigenschaften der Applikation als auch von den Merkmalen der Szene abhängig ist. Eine Lösung wäre für jede Applikation eine eigene räumliche Sortierung einzuführen. Diese Vorgehensweise würde aber einen enormen Verwaltungsaufwand hinsichtlich Speicher und Laufzeit nach sich ziehen [3]. Infolgedessen wird im Konzept der vorliegenden Arbeit ein Kompromiss geschlossen: Es gibt lediglich eine räumliche Repräsentation, die von allen Applikationen zu nutzen ist. Der Wert δ ist Teil der Beschreibung einer Szene, d.h. er wird vom Designer der Szene spezifiziert und zusammen mit den anderen Informationen gespeichert.

Da die Wahl der Kapazität δ Auswirkungen auf die Effizienz der einzelnen Anwendungen haben kann, ist ihre Einhaltung von großer Wichtigkeit. Innerhalb dynamischer Szenen fällt diese Aufgabe alles andere als einfach aus, denn bezüglich der Zellen eines Raumunterteiungsbaumes herrscht ein ständiges Kommen und Gehen der Elemente. Aus diesem Grund ist die erste Frage, wie die Anzahl der Aus- beziehungsweise Eintritte reduziert werden kann. Als Lösung verwenden viele Spiele für die Darstellung der statischen Elemente einen Raumunterteilungsbaum und für die Repräsentation der dynamischen Elemente eine gesonderte Datenstruktur. Eine derartige Vorgehensweise mag in Bezug auf das eingeschränkte Szenario eines Spiels ausreichend sein, hat aber im Falle generischer Szenen zwei große Nachteile. Zum einen ist die Anzahl der dynamischen Elemente beschränkt und zum anderen müssen Applikationen wie etwa Occlusion Cullings oder Kollisionserkennungen zwei Datenstrukturen bearbeiten. Insofern bleibt das Problem für massiv dynamische Szenen bestehen. Eine typische Form der Modellierung dynamischer Vorgänge liegt in dem Einsatz von Animationshierarchien, wie sie unter anderem in Abschnitt 6.3.3 behandelt werden. Derartige Hierarchien beschreiben nicht nur die Abhängigkeiten der Transformationen, sondern sie definieren auch die räumlichen Abstände der verbundenen Elemente zueinander. Üblicherweise sind diese Abstände sehr gering, d.h. die Elemente einer Animationshierarchie sind oftmals auch räumliche Nachbarn. Als Beispiel kann wieder der in Abbildung 6.11 illustrierte Greifarm dienen. Verlässt ein Element einer Animationshierarchie eine Zelle, so besteht eine hohe Wahrscheinlichkeit, dass auch die anderen Elemente der Hierarchie folgen. Somit impliziert ein einzelnes komplexes Element der Szene während eines Aus- oder Eintritts möglicherweise mehrere Verletzungen der Kapazität δ. Aus diesem Grund werden die Elemente einer Animationshierarchie mit Hilfe einer gemeinsamen Bounding Box gruppiert und repräsentiert. Abbildung 6.12 wirft dazu einen Blick auf die einzelnen Knotentypen des Raumunterteilungsbaumes. Innerhalb eines Baumes sind zwei grundsätzliche Arten von Knoten zu unterscheiden: Wohingegen die Blätter der Raumunterteilungsbäume die Elemente einer Szene repräsentieren, beschreiben die inneren Knoten die Zellen, in welche die Elemente einsortiert werden. Wie bereits in Abschnitt 6.3.1 erläutert, obliegt die Darstellung der Elemente einer Szene den Animationsagenten. Jeder Animationsagent ist in Form eines *Agentknoten* definiert, welcher unter anderem die visuelle Beschreibung des Agenten beinhaltet. Ähnlich den in Abschnitt 6.3.2 erläuterten Pools gibt es Unterschiede hinsichtlich Server und Client. Beispielsweise benötigt der Server eine Spezifikation des Verhaltens sowie eine Tabelle, welche die Clients identifiziert, auf denen der Animationagent momentan verfügbar ist. Analoges gilt für einen weiteren Blattknoten, nämlich dem *Hierarchieknoten*, der wie oben angedeutet eine

[3] Ganz abgesehen vom Kommunikationsaufwand aufgrund der Synchronisation der verteilten Datenstrukturen.

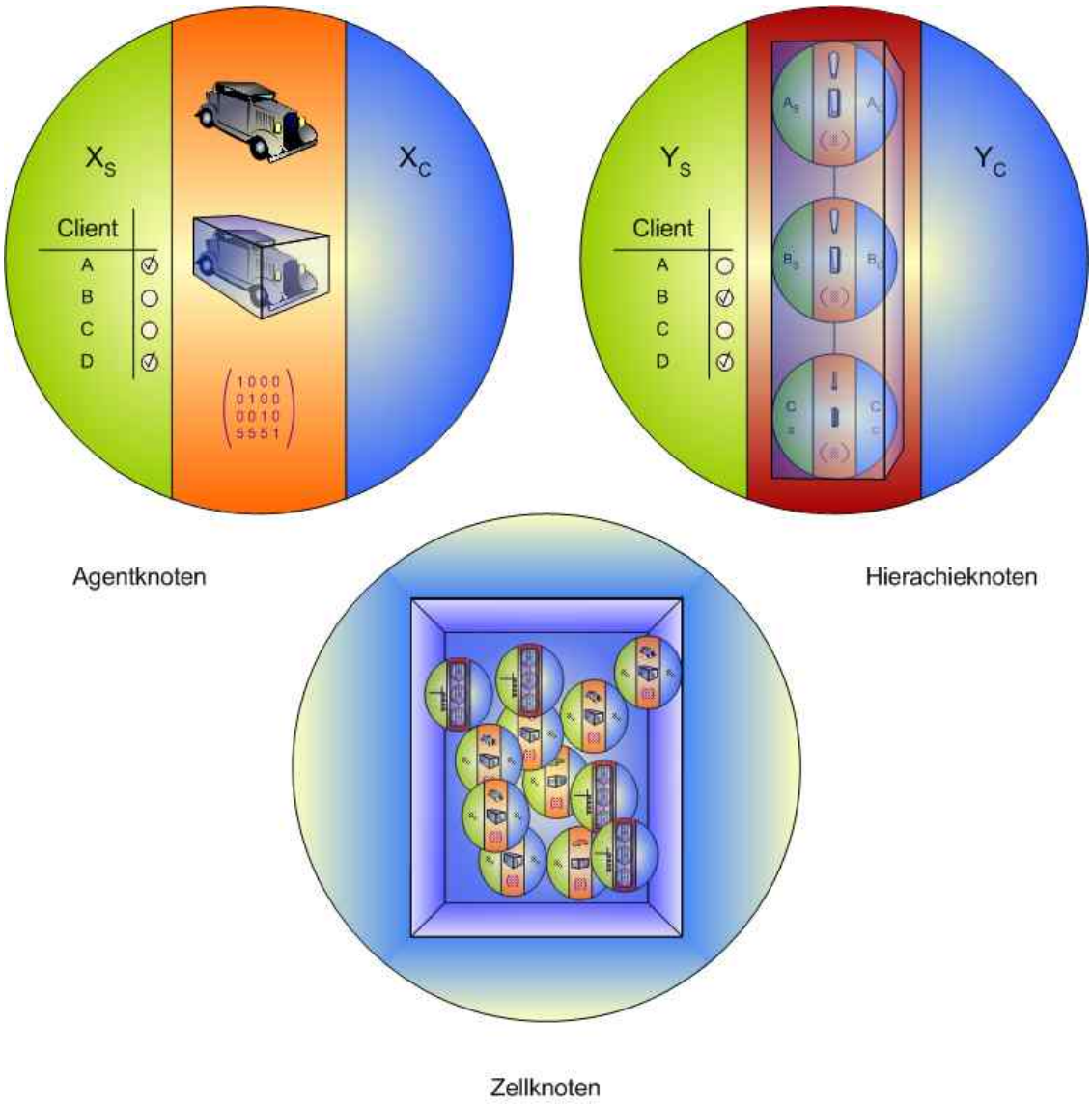

Abbildung 6.12: *Der Raumunterteilungsbaum besteht aus unterschiedlichen Knotentypen. Während ein einzelner Animationsagent durch einen Agentknoten repräsentiert wird, fasst ein Hierarchieknoten die Agenten einer Animationshierarchie unter einer gemeinsamen AABB zusammen. Agentknoten und Hierarchieknoten bilden die Blätter des Raumunterteilungsbaumes, wohingegen die Zellknoten den inneren Knoten des Baumes entsprechen. Sie beschreiben eine achsenparallele Region der Szene.*

Animationshierarchie von Agenten kapselt. Jeder *Zellknoten* beschreibt eine virtuelle Zelle des Raumunterteiungsbaumes. Alle Knoten verfügen über eine AABB, welche das Kriterium für die räumliche Zuordnung der Agentknoten und der Hierarchieknoten darstellt. Die AABB eines Zellknotens ist durch die Ausmaße der repräsentierten Zelle gegeben. Konsequenterweise kommen für die Unterteilung des Baumes lediglich achsenparallele Konzepte wie etwa k-D-Trees oder Octrees in Frage. Während die Geometrie der visuellen Beschreibung eines Agentknotens die AABB eines Agentknotens definiert, umschließt die AABB eines Hierarchieknotens die Geometrie aller beteiligter Agenten. Neben der Geometrie selbst ist auch deren Transformation aufgrund eventueller Animationen zu berücksichtigen. Weil letztere im

Animation Pool (siehe Abschnitt 6.3.3) vordefiniert sind, besteht die Option, die AABB eines Agentknotens oder Hierarchieknotens anhand der aktuellen Animation auszuwählen. Ein Agentknoten oder Hierarchieknoten wird einem Zellknoten nur dann als Kindknoten zugeordnet, wenn die AABB des Zellknoten die AABB der potentiellen Kandidaten vollständig beinhaltet. Andernfalls könnte es zu Ineffizienzen innerhalb von Suchvorgängen kommen. Da während eines Aus- oder Eintritts eines Hierarchieknoten lediglich dessen übergeordnete AABB als Kriterium herangezogen wird, kann der Hierarchieknoten auch nur eine Verletzung der Kapazität des betroffenen Zellknotens verursachen.

Für die weitere Reduzierung der Zellübertritte schlagen Sudarsky et al. [SG96] das Konzept der *Temporary Bounding Boxes* vor. Eine Temporary Bounding Box definiert für ein Element eine Region, welche das Element innerhalb eines bestimmten Zeitintervalls unter Garantie nicht verlässt. Demzufolge hängt die Größe der Bounding Box von den Transformationen des Elements während der spezifizierten Zeitspanne ab. Die Einordnung des Elements in den Raumunterteilungsbaum erfolgt nicht über seine AABB, sondern über die Temporary Bounding Box, wobei das betreffende Element während des vorgegebenen Zeitintervalls keine Verletzung der Kapazität δ verursacht. Nach dem Ablauf der Zeit muss im Gegensatz zum Konzept der Hierarchieknoten eine neue Temporary Bounding Box ermittelt werden. Ein Nachteil des Konzepts der Temporary Bounding Boxes ist, dass die resultierende Temporary Bounding Box eines Elements in der Regel wesentlich größer ausfällt als dessen AABB. Aus diesem Grund wird das Element in ein niedrigeres Level des Raumunterteilungsbaumes eingeordnet, wodurch die Effizienz vieler Anwendungen einen negativen Einfluss erhält. Beispielsweise steigt im Falle eines Occlusion Cullings die Wahrscheinlichkeit, dass ein Element anhand seiner Temporary Bounding Box als sichtbar identifiziert wird, obwohl es sich gar nicht im Sichtbereich des Betrachters befindet [4]. Zwar kann auch weiter die AABB des Elements verwendet werden, allerdings ist dies bereits ein zusätzlicher Test. Daher werden Temporary Bounding Boxes hauptsächlich auf Elemente außerhalb des Sichtbereichs angewendet, was bei einer Integration des Konzepts in die vorliegende Arbeit zwei schwerwiegende Nachteile mit sich bringen würde: Zum einen wäre die Verwendung der Raumunterteilungsbäume auf Sichtbarkeitsverfahren beschränkt und zum anderen könnte es lediglich auf Seite des Clients eingesetzt werden. Im Gegensatz zum Client hat der Server nämlich aufgrund der visuellen Areas-of-Interest (siehe Abschnitt 5.1.5) nicht nur einen, sondern viele Clients zu berücksichtigen. Als Fazit daraus bieten die Agentknoten und die Hierarchieknoten zwar einen weiteren Eintrag für eine achsenparallele Temporary Bounding Box an, allerdings bleibt deren Bestimmung als Option der Simulation überlassen.

Wie in Abschnitt 4.4 erläutert, impliziert das Überschreiten der Kapazität δ einer Zelle deren Unterteilung. Nach Abschnitt 5.1.1 können sich hierfür zwei der dort aufgeführten Operationen verantwortlich zeichnen, nämlich das Einfügen eines neuen Animationsagenten in die Szene sowie das Transformieren eines Agenten von einer Zelle in eine andere. Abbildung 6.13 zeigt, dass sich die Unterteilung bei entsprechender Anzahl der Agenten rekursiv fortset-

[4] Obgleich die größere AABB der Hierarchieknoten ebenfalls eine niedrigere Einsortierung impliziert, unterscheidet sich das Konzept der Hierarchieknoten in diesem Punkt von dem Konzept der Temporary Bounding Boxes. Aufgrund der räumlichen Nachbarschaft und dem durch die Animationshierarchie eingegrenzten Bewegungsradius ist die Wahrscheinlichkeit sehr hoch, dass sobald die AABB des Hierarchieknotens in den Sichtbereich kommt auch die betreffenden Animationsagenten sichtbar sind. Im Falle eines riesigen Raumschiffes, an dessen Bug und Heck sich jeweils Radargeräte bewegen, käme es aber auch hier zu Ineffizienzen.

zen kann. In dem Beispiel enthält die Zelle eines k-D-Trees links oben acht Kinderknoten, verfügt aber nur über eine Kapazität von fünf. Insofern erfolgt eine mittige Unterteilung, wobei die rechte resultierende Zelle immer noch sechs Animationsagenten beinhaltet. Sie muss daher weiter unterteilt werden bis zu dem rechts unten dargestellten Bild. Im Falle des Verschiebens eines Animationsagenten erhöht sich zwar die Anzahl der Agenten in der Zielzelle, andererseits reduziert sich aber auch die Anzahl der Agenten innerhalb der Ausgangszelle. Sowohl das Transformieren eines Agenten als auch die dritte in Abschnitt 5.1.1 aufgeführte Operation der Entfernung eines Agenten kann daher eine Unterschreitung der Kapazität δ mit sich bringen. In diesem Fall ist die Unterteilung wieder rückgängig zu machen. Abbildung 6.13 definiert hierzu als grundlegende Basisoperation die *Unite* Operation und vereinigt nach dem Entfernen zweier Agenten die betroffene Zelle wieder zu dem unten links dargestellten Bild. Die Unterteilung einer Zelle ist über die Basisoperation *Divide* abgedeckt. In der strengen Interpretation von Abbildung 6.13 impliziert das Unterschreiten der Kapazität δ die automatische Vereinigung der betroffenen Zellen. Um hier Operationen zu sparen, erfolgt die Definition der beiden Operationen nicht nur mit einer maximalen Kapazität δ_{max}, sondern auch mit einer minimalen Kapazität δ_{min}. Beide Kapazität spezifizieren also einen Toleranzbereich hinsichtlich der Agenten in einer Zelle.

Definition (Divide Operation): Überschreitet die Anzahl der Animationsagenten in einer Zelle C die Kapazität δ_{max}, so unterteilt die Divide Operation die Zelle C je nach Strategie in Unterzellen und fügt diese als Kindknoten in C ein. Die Animationsagenten in C werden den Unterzellen zugeordnet, deren AABB die AABB der Agenten vollständig umschließt.

Definition (Unite Operation): Unterschreitet die Anzahl der Animationsagenten in einem Teilbaum die Kapazität δ_{min}, so löscht die Unite Operation den gesamten Teilbaum bis auf die Wurzel des Teilbaumes. Alle verbleibenden Animationsagenten des Teilbaumes werden dessen Wurzel zugeordnet.

Die Verwendung der Hierarchieknoten und der optionalen Temporary Bounding Boxes bildet einen ersten Ansatz, um die Zahl der Divide und Unite Operationen zu reduzieren. In beiden Fällen beruht die Idee auf der Minderung der Grenzverletzungen, sofern das Verlassen beziehungsweise Betreten einer Zelle von Seiten eines Animationsagenten als solche erachtet wird. Nichtsdestotrotz implizieren sowohl die Hierarchieknoten als auch die Temporary Bounding Boxes eine Reorganisation des Raumunterteilungsbaumes, sobald eine Verletzung des durch δ_{min} und δ_{max} definierten Toleranzbereiches auftritt. Die Ursache hierfür liegt darin begründet, dass beide Ansätze lediglich lokale Kriterien darstellen, d.h. sie berücksichtigen ausschließlich den Bewegungsvorgang eines einzelnen beziehungsweise einer eingeschränkten Gruppe von Animationsagenten und nicht die Transformationen aller Agenten einer Szene. In vielen Szenarien wie etwa Städtemodellen treten aber Situationen auf, wie sie in Abbildung 6.14 illustriert sind: Ein Animationsagent A verlässt eine Zelle C_1 und betritt eine Zelle C_2, die gerade von Agent B geräumt wird. B wiederum betritt die Zelle C_3, während Agent C von Zelle C_3 in die ursprüngliche Zelle von Agent A, also C_1, überwechselt. In einem einfacheren Beispiel tauschen zwei Animationsagenten lediglich die von ihnen beanspruchten Zellen. Jede Bewegung für sich betrachtet kann eine Verletzung der Kapazität der betroffenen Zellen und somit eine Reorganisation des Raumunterteilungsbaumes verursachen. Am Ende aller Transformationen entspricht die Struktur des Raumunterteilungsbaumes jedoch der Ausgangssituation [SS04]. In einem kontinuierlichem Zeitmodell ist es sehr

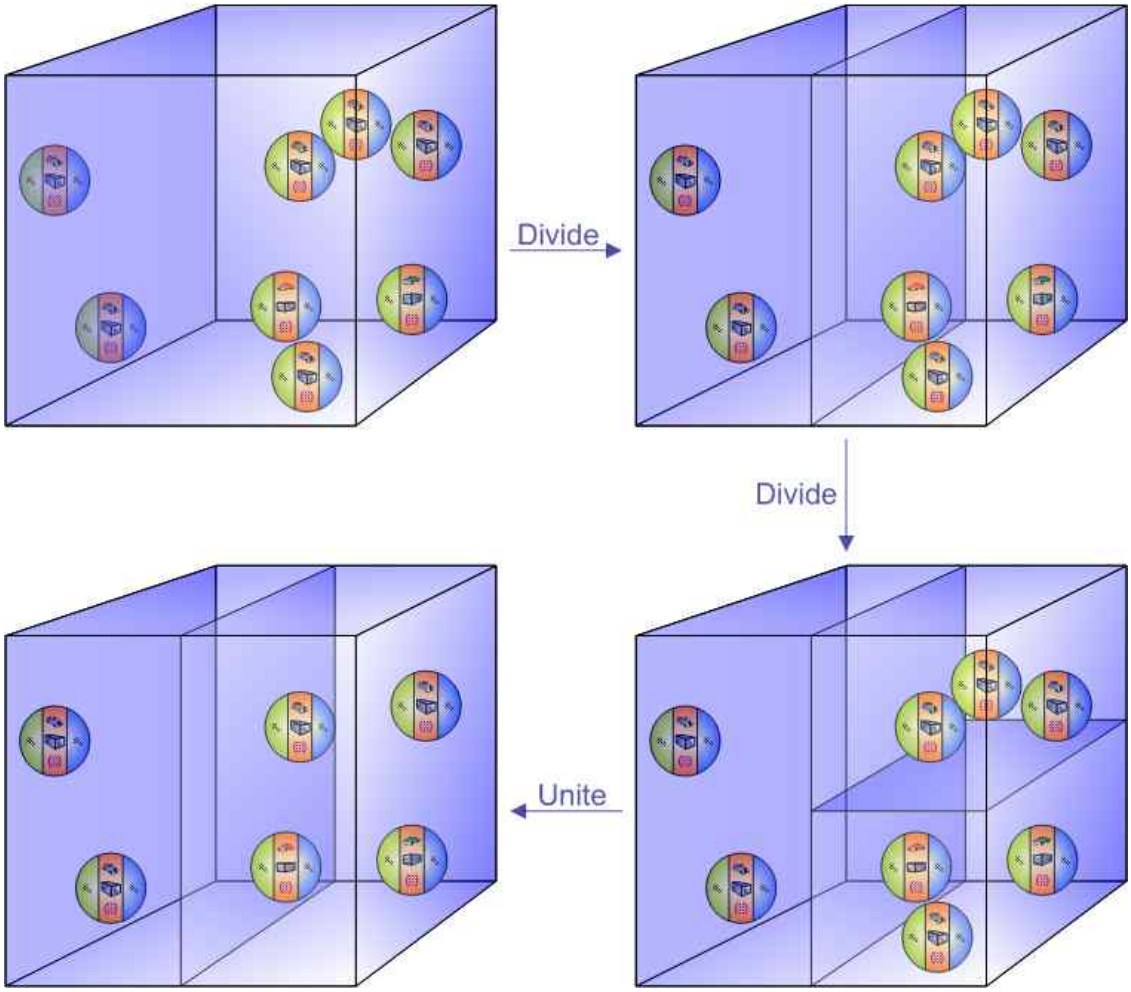

Abbildung 6.13: *Die Reorganisation des Raumunterteilungsbaumes basiert auf den beiden Basisoperationen Divide und Unite. Erstere unterteilt eine Zelle, sofern deren maximale Kapazität überschritten wird. Die Unite Operation fasst einen gesamten Teilbaum wieder zu einem einzelnen Knoten zusammen, falls die Zahl der Animationsagenten im Teilbaum unter die minimale Kapazität sinkt.*

unwahrscheinlich, dass derartige, sich gegenseitig kompensierende Bewegungen alle zum exakt identischen Zeitpunkt auftreten. Anders sieht es aber in einem diskreten Zeitmodell aus, wie es glücklicherweise in Simulationen häufig anzutreffen ist. Hier kann immer das durch zwei aufeinander folgende Zeitschritte definierte Zeitintervall betrachtet und die kompensierenden Bewegungen evaluiert werden. Als Konsequenz wird während des Zeitintervalls nicht mehr jede Transformation für sich getrennt berücksichtigt, sondern nur noch die resultierenden Verletzungen des Toleranzbereiches aufgrund der Bewegung aller Elemente. Ein weiterer Vorteil des Konzept sich gegenseitig ausgleichender Bewegungen liegt in der Anwendbarkeit auf alle drei der in Abschnitt 5.1.1 geforderten Operationen, also dem Einfügen eines Animationsagenten in die Szene sowie dessen Entfernen beziehungsweise Transformieren innerhalb der Szene.

Da die Berücksichtigung eines einzelnen Zeitpunktes nun nicht mehr ausreichend ist, muss die Idee, die Animationsagenten einfach im korrespondierenden Zellknoten zu verwalten, neu überdacht werden. Abbildung 6.15 zeigt stattdessen eine Dreiteilung der Zellknoten. Die

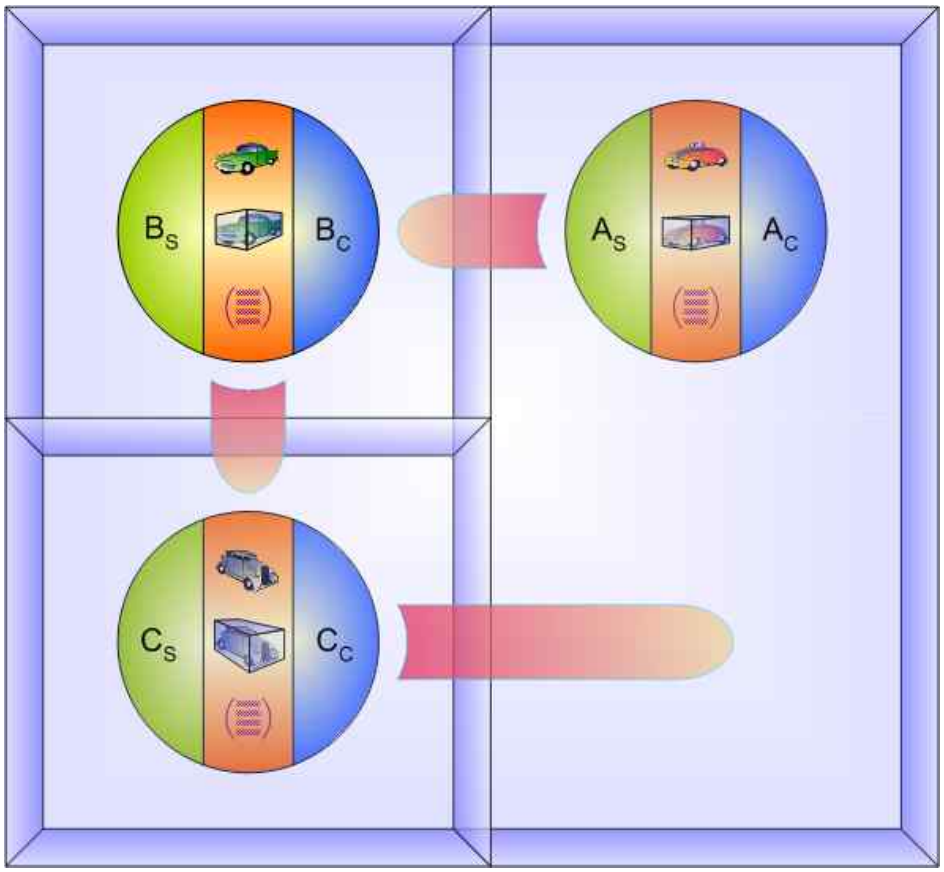

Abbildung 6.14: *Die Anzahl der bei der Reorganisation anfallenden Basisoperationen kann durch die Berücksichtigung sich gegenseitig kompensierender Bewegungen deutlich reduziert werden.*

obere Hälfte des Knotens, der sogenannte *Current Kontainer*, repräsentiert den aktuellen Zeitschritt t_i der Simulation und beinhaltet die momentan in der Zelle vorhandenen Animationsagenten. Die Anzahl dieser Agenten wird im folgenden mit der Funktion $n_{cur}(X,t)$ beschrieben, wobei X die Zelle und t den jeweiligen Zeitschritt repräsentiert. Die beiden Kontainer in der unteren Hälfte des Knotens beziehen sich auf das Zeitintervall zwischen t_i und dem nächsten Zeitschritt t_{i+1} der Simulation. Wohingegen der *Out Kontainer* auf die Agenten verweist, welche die Zelle innerhalb des Zeitintervalls verlassen, referenziert der *In Kontainer* die Agenten, welche die Zelle während besagter Zeitdauer betreten. Das Eintragen der Agenten in die entsprechenden Kontainer geschieht von Seiten der Simulation. Die Funktionen $n_{in}(X,t)$ und $n_{out}(X,t)$ sind analog zu $n_{cur}(X,t)$ zu verstehen, allerdings mit dem Unterschied, dass t das Zeitintervall vom aktuellen Zeitschritt bis zum nächsten umfasst. Sobald die Simulation den Zeitschritt t_{i+1} erreicht, erfolgt eine Analyse der einzelnen Kontainer. Die Differenz $n_{in}(W,t_i) - n_{out}(W,t_i)$ beschreibt ein lokales Maß für die Änderungen innerhalb einer Zelle W während des Zeitintervalls t_i bis t_{i+1}. Der Wert $n_{cur}(W,t_{i+1})$ mit

$$n_{cur}(W,t_{i+1}) = n_{cur}(W,t_i) + n_{in}(W,t_i) - n_{out}(W,t_i) \tag{6.2}$$

beschreibt die Anzahl der Animationsagenten im Current Container einer Zelle W für den

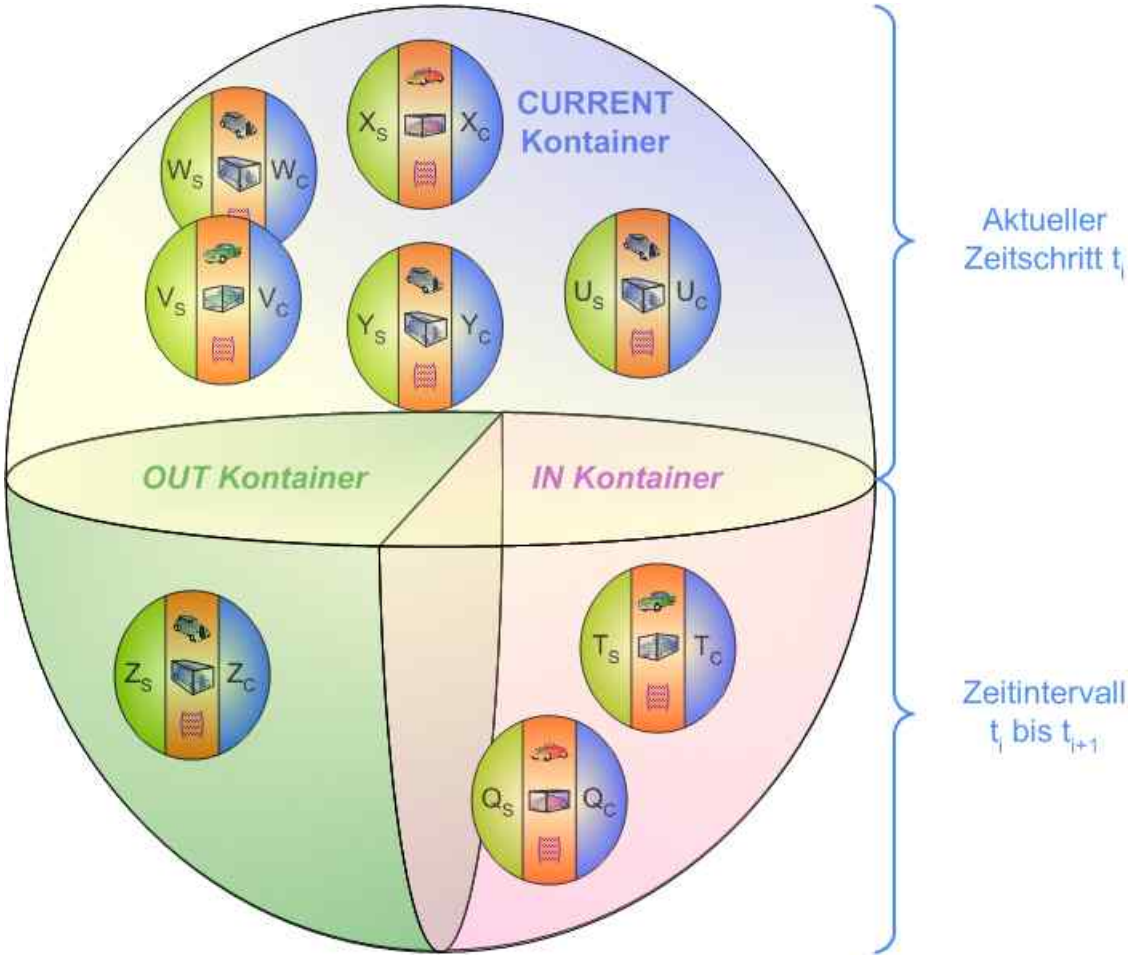

Abbildung 6.15: *Aufgrund des Konzepts der kompensierenden Bewegungen ist die Betrachtung eines Zeitintervalls erforderlich, das durch die Zeitschritte der Simulation vorgegeben wird. Während der In und der Out Container die Änderungen für den nächsten Zeitschritt repräsentieren, beschreibt der Current Container den aktuellen Zustand. Letzterer ist bis auf die kurze Phase der Reorganisation bei Beginn eines neuen Zeitschritts immer konsistent und erlaubt parallele Lesezugriffe.*

Zeitschritt t_{i+1}. Das alleine ist aber nicht ausreichend, weil die Unite Operation einen ganzen Teilbaum in Betracht zieht. Sei nun U ein Teilbaum mit W als Wurzel und die Zellen $C_1, \ldots, C_k$ die in U enthaltenen Knoten [5]. Dann kann für U die Anzahl der Animationsagenten $n_{cur}(U, t_{i+1})$ im Current Container hinsichtlich des Zeitschritts t_{i+1} folgendermaßen vorherberechnet werden:

$$n_{cur}(U, t_{i+1}) = \sum_{j=1}^{k} (n_{cur}(C_j, t_i) + n_{in}(C_j, t_i) - n_{out}(C_j, t_i)) \tag{6.3}$$

Das Erreichen des Zeitschritts t_{i+1} läutet eine kurzfristige Reorganisation des Raumunterteiungsbaumes ein. Innerhalb dieser Phase werden für alle von Veränderungen betroffenen Zellen die Animationsagenten des Out Kontainers aus dem Current Kontainer entfernt und

[5] Die Wurzel ist Teil der Knoten.

die Agenten des In Kontainers in den Current Kontainer einzutragen. Vor der Leerung des In und Out Kontainers einer Zelle W wird mit Hilfe der Gleichungen 6.2 und 6.3 analysiert, ob eine der beiden Basisoperationen Divide und Unite auf W anzuwenden ist. Die Überprüfung basiert auf den beiden nachfolgend aufgeführten Kriterien. Sind beide Kriterien nicht gegeben, so ist eine Bearbeitung der Zelle überflüssig.

- Ist W nicht unterteilt und es gilt $n_{cur}(W, t_{i+1}) > \delta_{max}$, dann muss die Divide Operation auf W angewendet werden.
- Ist W unterteilt und es gilt $n_{cur}(U, t_{i+1}) < \delta_{min}$, wobei U den Teilbaum mit W als Wurzel beschreibt, so muss die Unite Operation auf W angewendet werden.

Neben der Berücksichtigung kompensierender Transformationen bietet das in Abbildung 6.15 dargestellte Konzept der Zellknoten einen zusätzlichen positiven Nebeneffekt. Während des Zeitintervalls von t_i bis t_{i+1} bleiben die Current Container aller Zellknoten unverändert und stellen somit eine konsistente Beschreibung des momentanen Zustands der Szene dar. Aus diesem Grund können abgesehen von der kurzen Reorganisation zu Beginn eines Zeitschritts parallele Lesezugriffe auf die Current Kontainer erfolgen, was aufgrund der massiven Kommunikation zwischen der Szenen-Komponente und den übrigen Komponenten einen enormen Geschwindigkeitszuwachs bedeutet (siehe Abschnitt 6.2). Schreibzugriffe werden über die In Kontainer und Out Kontainer abgefangen und wirken sich erst auf den nächsten Zeitschritt aus. Dadurch ist zum Beispiel auf Seite des Clients das Einfügen neuer Animationsagenten durch die Demultiplexer-Komponente in die Szenen-Komponente möglich, obgleich die Präsentation-Komponente gerade die Szene visualisiert.

Die zuvor ausgearbeiteten Konzepte der Hierarchieknoten, der Temporary Bounding Boxes sowie der kompensierenden Transformationen sind unabhängig von der Strategie eines Raumunterteilungsbaumes einsetzbar. Die Wahl der Strategie hat maßgeblichen Anteil an der Anzahl der Schnittelemente und damit auch auf die Menge permanenter Verletzungen der Kapazität δ. Hierunter sind Situationen zu verstehen, in denen ein Schnittelement keiner Kinderzelle einer Zelle C zugeordnet werden kann, obgleich die Kapazität von C bereits überschritten ist. Als Beispiel sei an die in Abschnitt 4.4 beschriebenen Lösungsvorschläge von Greene et al. [GKM93] für den Umgang mit Schnittelementen erinnert: Die einzige bezüglich dynamischer Szene geeignete Vorgehensweise ist, die Schnittelemente einfach in der Zelle zu belassen, deren AABB die Schnittelemente vollständig umfasst. Im Zweifelsfall ist dies die Wurzel des Raumunterteilungsbaumes. Angenommen ein nahezu planares Stadtmodell ist auf Höhe der $x - z$ Hyperebene definiert und soll mit Hilfe eines Octrees räumlich repräsentiert werden. Dann kommt es zu keiner Unterteilung, sondern alle Elemente der Szene fallen direkt der Wurzel des Octrees zu. Aus Sicht der Reorganisation sind derartig extreme Verletzungen des Toleranzbereiches δ_{min} bis δ_{min} nicht weiter tragisch. Im Gegenteil, es führt sogar zu einer Vereinfachung des Problems, da etwa im Falle des Stadtmodells überhaupt keine Reorganisation mehr nötig wäre. Dies ist aber nicht im Sinne der Anwendungen, deren Effizienz von einer geeigneten räumlichen Sortierung der Szene abhängig ist.

Wie bereits in Abschnitt 5.1.1 erwähnt, bietet der Ansatz der Nine-Area-Trees von Chang et al. [CLC03] eine elegante Lösung für das Problem der Schnittelemente. Abbildung 6.16

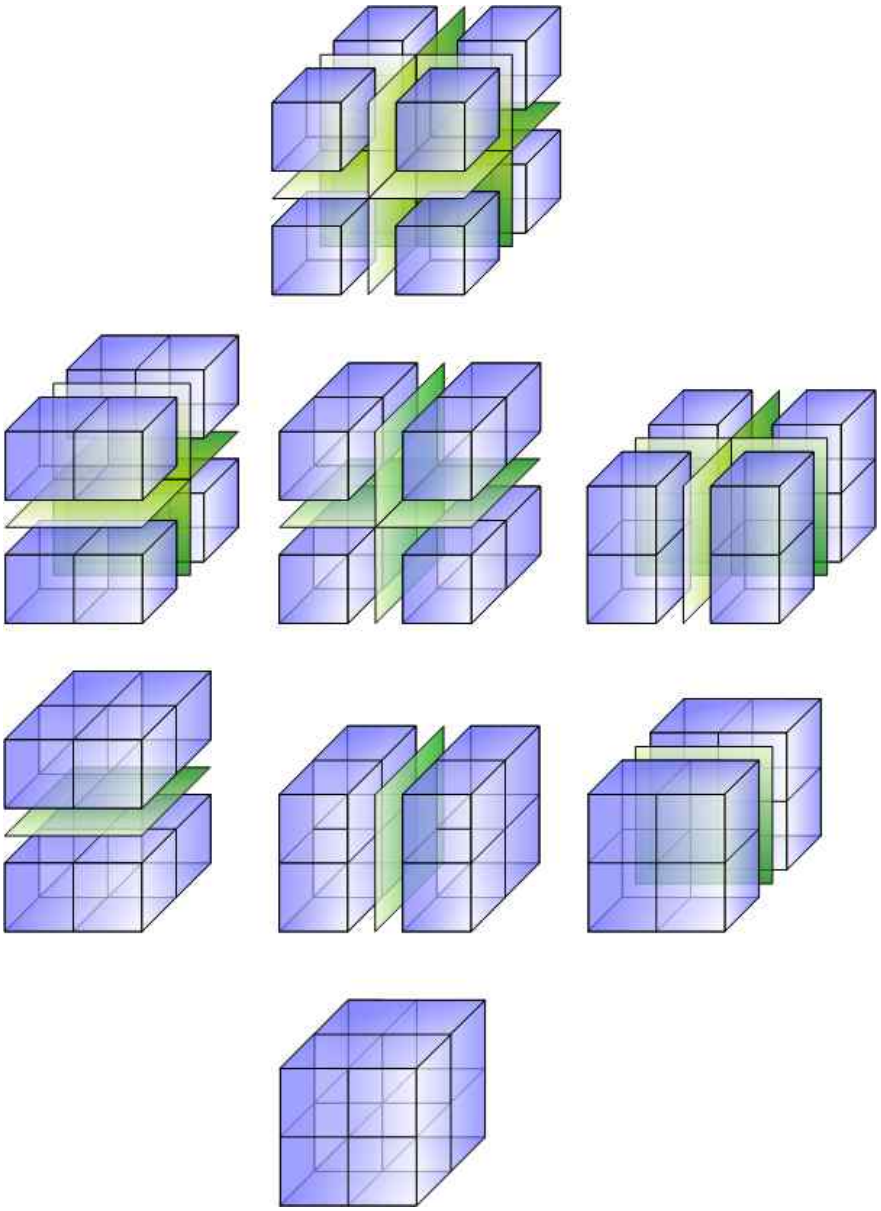

Abbildung 6.16: *Die 27 möglichen Regionen des Raumunterteilungsbaumes ergeben sich alle aus den Octanten. Schneidet ein Animationsagent keine der drei Hyperebenen, so wird er einem der Octanten in der obersten Reihe zugeordnet. Die folgenden Reihe entsprechen dem Schneiden einer, zweier sowie dreier Hyperebenen. Animationsagenten, welche der untersten Variante zugesprochen werden, müssen sich im Ursprung der Zelle schneiden.*

illustriert die Übertragung des Konzepts auf den dreidimensionalen Raum, weshalb anstelle des ursprünglichen Quadtrees ein Octree das Grundgerüst des Raumunterteilungsbaumes darstellt. Sobald eine Zelle C aufgrund des Überschreitens der Kapazität δ_{max} unterteilt werden muss, erfolgt zuerst ein Test auf eine mögliche Zuordnung der in der Zelle vorhandenen Animationsagenten in einen der acht Octanten. Als Bedingung für einen erfolgreichen Test darf ein Animationsagent keine der drei achsenparallelen Hyperebenen der Zelle C schneiden. Im Falle eines negativen Ergebnisses ist die nächstbeste Möglichkeit, den Agenten in ein Octantenpaar einzutragen. Abbildung 6.16 zeigt die entsprechenden Fälle in der zweiten

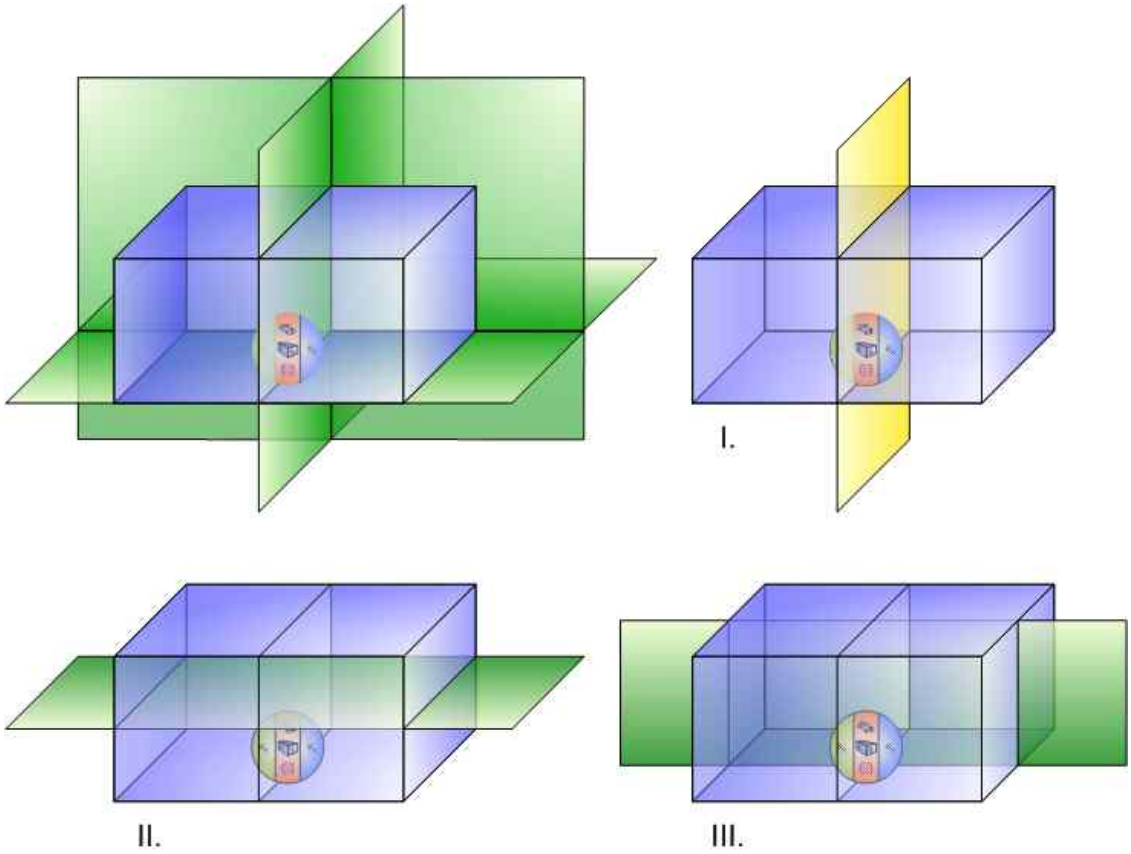

Abbildung 6.17: *Wird ein Animationsagent aufgrund des Schneidens einer oder mehrerer Hyperebenen einer Tochterzelle zugeordnet, die sich über mehrere Octanten erstreckt, so sind die Unterteilungsoptionen der Tochterzelle entsprechend eingeschränkt.*

Reihe. Konsequenterweise darf ein Animationsagent hierfür maximal eine der Hyperebenen der Zelle C schneiden. Fällt auch dieser Test negativ aus, d.h. der Animationsagent schneidet mindestens zwei der Hyperebenen, so kommen nun die in Abbildung 6.16 dargestellten Bereiche der dritten Reihe mit jeweils vier Octanten in Frage. Ein weiterer Fehltest impliziert, dass der Agent alle drei Hyperebenen schneidet und somit im Ursprung der Zelle liegt. Zudem müssen sich sämtliche Animationsagenten mit dieser Eigenschaft in mindestens einem gemeinsamen Punkt, nämlich dem Ursprung der Zelle, schneiden. Die Wahrscheinlichkeit für eine große Zahl derartiger Fälle ist in üblichen Szenen äußerst gering. Aus diesem Grund können die betreffenden Animationsagenten direkt der Zelle C zugewiesen werden, ohne massive Verletzungen der Kapazität δ_{max} befürchten zu müssen.

Sobald ein Animationsagent in eine der Zellen der zweiten beziehungsweise dritten Reihe einzuordnen ist, erfolgt dort eine einmalige Sortierung nach dem Muster eines k-D-Trees. Ein entscheidender Faktor für die Effizienz eines k-D-Trees ist die Wahl der Hyperebene hinsichtlich der Unterteilung, welche nach Abbildung 6.17 diversen Einschränkungen unterliegt. Der Grund für die Einsortierung eines Animationsagenten in eine Zelle O aus zwei Octanten ist dessen Schneiden einer Hyperebene h. Überschreitet die Anzahl der Agenten auch die Kapazität dieser Zelle, so ist die Verwendung der Hyperebene h für die Unterteilung ungeeignet. Stattdessen kommen gemäß Abbildung 6.17 bei einem k-D-Tree mit einer mit-

telnden Unterteilungsstrategie lediglich die Fälle A und B in Frage. Rekrutiert die Zelle O gar aus vier Octanten, so ist die Wahl der Hyperebene auf eine beschränkt, da die Animationsagenten die beiden anderen Hyperebenen der Zelle C schneiden. Bei weiteren rekursiven Unterteilungen innerhalb des Teilbaumes mit O als Wurzel stimmt die Zuordnungstrategie der Animationsagenten wieder mit der Vorgehensweise des Octrees überein.

In sehr extremen Fällen hat nach dem zuvor beschriebenen Konzept eine Zelle bis zu 27 Kindknoten zu verwalten. Auf der einen Seite ergibt sich daraus ein nicht zu vernachlässigender Speicherbedarf pro Zelle, auf der anderen Seite aber eine ganze Reihe von Vorteilen:

- Die Animationsagenten sind deutlich tiefer in dem Raumunterteilungsbaum eingeordnet als im Falle eines Octrees mit der von Greene et al. beschriebenen Methode. Als Konsequenz gibt es eine wesentlich geringere Zahl an permanenten Verletzungen der Kapazität δ_{max}. Damit sinkt die Wahrscheinlichkeit, während einer Traversierung des Baumes den erwähnten teuren Elementtest durchführen zu müssen.

- Mit dem Einsortieren in tiefere Level sinkt das Volumen der Zellen, die somit eines bessere Approximation an die enthaltenen Animationsagenten darstellen. Hierdurch steigt die Effizienz vieler Algorithmen wie etwa Occlusion Culling Verfahren oder Kollisionserkennungen.

- Da die Kapazität der Zellen weitestgehend eingehalten wird, kann für jeden unterteilten Zellknoten eine weitere AABB bestimmt werden, welche die AABBs der in der Zelle vorhandenen Animationsagenten exakt umschließt. Hierbei ist zu bedenken, dass es sich im Falle unterteilter Zellknoten nur um Animationsagenten handeln kann, die sich im Ursprung der Zelle schneiden. Aus diesem Grund sollte üblicherweise sowohl die Anzahl der Animationsagenten als auch die resultierende AABB massiv eingeschränkt sein. Die geringe Zahl der Agenten ermöglicht die Bestimmung der AABB auch in dynamischen Szenen. Die Berechnung einer entsprechenden AABB für alle Zellen hängt von der Größe der Kapazität δ_{max} ab.

Ein Bearbeitung aller 27 Kindzellen ist beispielsweise im Falle eines Occlusion Culling Verfahrens eine kostspielige Angelegenheit. Weiterhin besteht ähnlich zu den in Abschnitt 4.5 beschriebenen Bounding Volume Hierarchien eine Überlappung zwischen den Kindzellen. Glücklicherweise fällt diese Überlappung exakt deckungsgleich aus, weshalb die Ergebnisse für Bereiche aus mehreren Oktanten mit Hilfe der Resultate der einzelnen Oktanten geschlussfolgert werden können. Sind zwei Oktanten unsichtbar, so ist auch die beide Oktanten beanspruchende Kindzelle O unsichtbar. Insofern ergibt sich hier im Vergleich zu einem Octree kein Mehraufwand. Sind zwei Oktanten dagegen sichtbar, so gilt gleiches auch für O. Der Test alleine stellt noch keinen Zeitfaktor dar, aber nun beschreiben sowohl die Oktanten als auch die Zelle O die gleiche Region. Derartige Überlappungen können sich negativ auf die Effizienz der Anwendungen auswirken. Jedoch greift in vielen Fällen die Berechnung der exakten AABB, wie sie im letzten Punkt der obigen Aufzählung erläutert ist.

Eine Zuordnung der Animationsagenten in eine der 27 Kindzellen ist ebenfalls ohne großen Aufwand zu bewerkstelligen. Hierzu muss lediglich die AABB eines Agenten in Relation zum

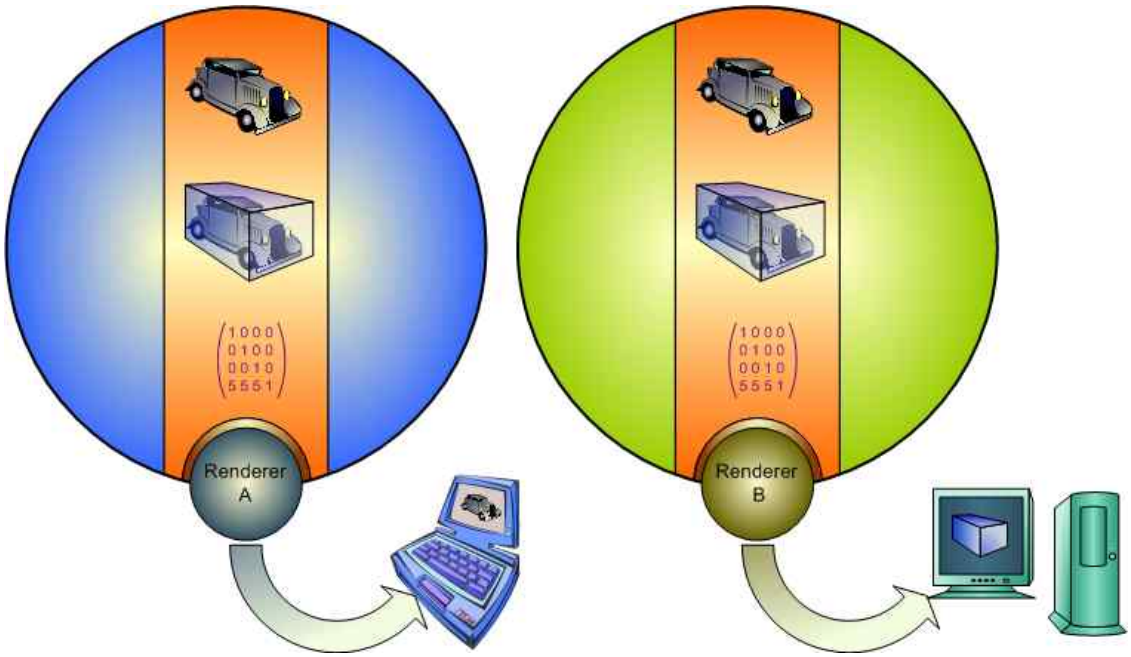

Abbildung 6.18: *Renderer sind Bausteine der lokalen Plattform und werden mit Empfang eines Animationsagenten in selbigen eingesetzt, um beispielsweise die Visualisierung des Agenten zu ermöglichen. Sie erlauben nicht nur die Kapselung plattformabhängiger Merkmale, sondern zusätzlich ein beliebiges Level-of-Abstraction.*

Ursprung der Zelle gesetzt werden. Bei geschickter Indizierung der Kindzellen können die Ergebnisse des Vergleichs direkt in eine Referenz auf den korrespondierenden Kindknoten umgewandelt werden. Der entsprechende Algorithmus ist zusammen mit der Reorganisation dynamischer Raumunterteilungsbäume in Abschnitt 7.3.3 erklärt.

6.3.5 Das Konzept der Renderer

Innerhalb der vorliegenden Arbeit stellen Renderer Komponenten dar, welche eine Datenstruktur traversieren, um eine bestimmte Form der Ausgabe zu erzeugen [SBS03]. Ähnlich dem Begriff des Renderns ist unter einer Ausgabe nicht ausschließlich die Visualisierung einer Szene zu verstehen. Vielmehr können es beliebige Informationen sein, die von anderen Komponenten angefordert werden. Die Idee hinter dem Ansatz der Renderer ist sowohl die Kapselung systemspezifischer Eigenschaften als auch die unterschiedliche Interpretation identischer Datensätze. Ziel ist dabei nach Abschnitt 5.1.1 die Bereitstellung einer Schnittstelle, welche den transparenten Austausch der Renderer ermöglicht. Abbildung 6.18 zeigt ein Server- und ein Client-System, auf dem jeweils der gleiche Animationsagent visualisiert wird. Während Renderer A auf dem Client eine Visualisierung der Geometrie des Modells anbietet, erzeugt Renderer B lediglich ein Level-of-Abstraction in Form einer Bounding Box. Beide Renderer sind jeweils Bestandteil ihrer lokalen Plattform und werden bei Bedarf dem Animationsagenten dynamisch eingefügt.

Die Szenen-Komponente enthält drei grundlegende Datenstrukturen zur Beschreibung einer Szene, welche in den vorhergehenden Abschnitten erläutert wurden. Bei diesen Strukturen

handelt es sich um den Elementgraphen, die Pools und den Raumunterteilungsbaum. Entsprechend existieren drei verschiedene Arten von Renderer, nämlich die *Elementgraphrenderer*, die *Poolrenderer* und die *Raumrenderer*. Die Schnittstelle zwischen Renderern und Datenstrukturen basiert auf der Bereitstellung von Methoden, die eine Traversierung der Strukturen erlauben. Elementgraph und Raumunterteilungsbaum sind jeweils hierarchische Bäume, insofern sind hier Optionen gefordert, die zum Beispiel die Deepest First Traversierung ermöglichen. Pools dagegen entsprechen zumindest nach außen hin linearen Strukturen.

In einigen Fällen kann die gegenseitige Verwendung der Renderer erforderlich sein. Beispielsweise ist es denkbar, dass ein Raumrenderer den Raumunterteilungsbaum anhand eines Occlusion Culling Verfahrens traversiert und für jeden sichtbaren Animationsagenten einen Elementgraphrenderer initiiert.

6.4 Die Progressive-Generator-Komponente

Die Progressive-Generator-Komponente ist ein Beispiel für einen Elementgraphrenderer. Ihre Aufgabe ist nach Abschnitt 6.1 die Umwandlung der visuellen Informationen eines Animationsagenten in ein entsprechendes progressives Format. Ziel ist nicht nur die Verwendung der Komponente im Rahmen eines Vorverarbeitungsschritts, sondern die Konvertierung interaktiv manipulierter Modelle zur Laufzeit. Dadurch kann der Benutzer auf Seite des Clients ein Modell bearbeiten und anschließend dem Server zukommen lassen, welcher das Resultat in die Szene einpflegt und mit den Clients synchronisiert.

Vor der eigentlichen Konvertierung eines Modells traversiert die Komponente dessen Elementgraph und setzt analog zum Konzept der Szenegraphen (siehe Abschnitt 4.7) den aktuellen Status des jeweils besuchten Knotens. Hierfür wird der zu einem Knoten korrespondierende Pooleintrag vermerkt. Enthält ein Pfad innerhalb des Elementgraphen identische Knotentypen, so überschreibt der zuletzt besuchte Knoten den Zustand seiner vorangehenden Artgenossen. Dies gilt auch im Falle von per Vertex beziehungsweise per Facet Knoten, d.h. ein *Color Facet Node* ersetzt den Status eines *Color Array Node*. Mit dem Erreichen eines Topologyknotens beginnt die Umwandlung in das progressive Format. Der nun vorhandene Gesamtzustand wird im folgenden auch als *Dependency* bezeichnet. Der Grund für diese Bezeichnung lässt sich anhand Abbildung 6.6 erahnen: Eine Dependency repräsentiert den bis zum Topologieknoten ermittelten Zustand unter Berücksichtigung aller Zusatzattribute. Zwar ist der Topologieknoten ein eigenständiger Knoten, dennoch muss er den Anforderungen seiner Vorgänger genügen. Beispielsweise darf er nicht mehr Scheitelpunkte indizieren als in dem aktuellen *Vertex Array Node* angegeben.

Das Rückgrat der progressiven Umwandlung bildet ein Simplifizierungsalgorithmus, welcher jeweils eine Dependency zur Verarbeitung entgegen nimmt. Das Konzept der Simplifizierung basiert ausschließlich auf dem Half-Edge Collapse Operator (siehe Abschnitt 4.3.1). Somit werden im Gegensatz zu Hoppe [Hop96] keine neuen Scheitelpunkte bestimmt, sondern lediglich die topologischen Informationen vereinfacht. Die Berechnung neuer Scheitelpunkte kann zwar zu einer verbesserten Approximation an das Originalmesh $\hat{M}$ führen, erfordert aber sowohl bei der Simplifzierung auf Seite des Servers als auch bei der Verfeinerung auf

Seite des Clients einen höheren Rechenaufwand. Aufgrund der Unterstützung schwächerer Endgeräte wie etwa PDAs ist insbesondere der letzte Punkt ein entscheidender Faktor.

Der Forderung nach Geschwindigkeit fällt nicht nur die Bestimmung neuer Informationen wie etwa besagter Scheitelpunkte zum Opfer, sondern auch die Einhaltung möglicher Fehlerschranken. Mit jedem Scheitelpunkt eines Dreiecksnetzes ist ein Fehler assoziiert, der ein Maß für die durch das Entfernen des betreffenden Punktes verursachte schlechtere Approximation an das Originalmesh beschreibt. Der Fehler selbst wird durch eine lokale Fehlermetrik bestimmt, welche den korrespondierenden Keil eines Scheitelpunktes in Betracht zieht. Ein Entfernen des Scheitelpunktes impliziert also nicht nur einfach eine geringere Approximation an das Originalmesh, sondern erfordert die Neuberechnung des Fehlers der im Keil vorhandenen Scheitelpunkte. Bei einer entsprechenden Metrik kann davon sogar noch die weitere Umgebung des Keils betroffen sein. Wie bereits in Abschnitt 4.3.1 angedeutet, ist vor allem die Einhaltung der globalen Fehlerschranke äußerst aufwändig. Sie stellt ein Kriterium dar, wann der Vorgang der Simplifizierung zu beenden ist, nämlich sobald eine gewisse Qualität der Approximation unterschritten wird. Konsequenterweise sind beim Vernachlässigen von Fehlerschranken schlechtere visuelle Ergebnisse als etwa im Falle von Klein et al. [KLS96] möglich, dafür aber bedeutend geringere Laufzeiten zu erwarten. Zudem ist nach Anforderung 2.3 die Adaptivität der Daten an die Leistungsmerkmale der Endgeräte verlangt, d.h. als Kriterium ist nicht die Güte der Approximation, sondern die exakte Vorgabe der zu übertragenden Dreiecke ausschlaggebend. Damit die letztlich transferierten Dreiecke auch bei einer geringen Anzahl eine möglichst gute Annäherung darstellen, zeichnet sich vor allem die verwendete Metrik zur Bestimmung des Fehlers eines Scheitelpunktes verantwortlich.

Ähnlich zu Schröder et al. [Sch97] verändert die Simplifizierung die Topologie eines Dreiecksnetzes, weshalb sich während des Vorgangs mehrere Zusammenhangskomponenten bilden können. Die Intention ist allerdings, die Topologie so lange wie möglich aufrecht zu erhalten und erst gegen Ende auch entsprechende Bereiche eines Meshes zu bearbeiten. Hierdurch wird eine für niedrige Bandbreiten und schwächere Endgeräte günstigere Kompression erreicht, ohne hinsichtlich leistungsfähigerer Geräte gleich eine massive Qualitätsminderung in Kauf nehmen zu müssen.

Die Simplifizierung beginnt mit der Berechnung des Fehlers ϵ_v für jeden Scheitelpunkt v des Dreiecksnetzes. Dabei wird als erstes der zu v korrespondierende Keil ermittelt und dieser auf die von Schröder et al. klassifizierten Sonderfälle überprüft. Keile, die entweder am Rand des Dreiecksnetzes liegen oder eine Auftrennung des Polygonnetzes in mehrere Zusammenhangskomponenten implizieren, erhalten einen maximalen Fehler. Abbildung 6.19 illustriert die Identifizierung der beiden Situationen anhand eines zum Scheitelpunkt v_0 gehörenden Keils. Ausgehend von einem beliebigen, zu v_0 adjazenten Scheitelpunkt wird versucht, innerhalb des Keils einen v_0 ausschließenden Zyklus zu finden, d.h. einen Weg der zum Ausgangspunkt zurückkehrt, ohne v_0 zu besuchen. Im Falle des linken Bildes kann der Ausgangspunkt nicht wieder erreicht werden, weshalb es sich hier um einen Rand handelt. Obgleich auf der rechten Seite ein entsprechender Zyklus existiert, bleibt ein Dreieck von dem beschrittenen Weg unberücksichtigt. Ein Entfernen von v_0 würde in dieser Situation das Auseinanderreißen des Dreiecksnetzes in zwei Zusammenhangskomponenten verursachen.

Durch die Markierung mit einem maximalen Fehlerwert werden derartige Keile beziehungs-

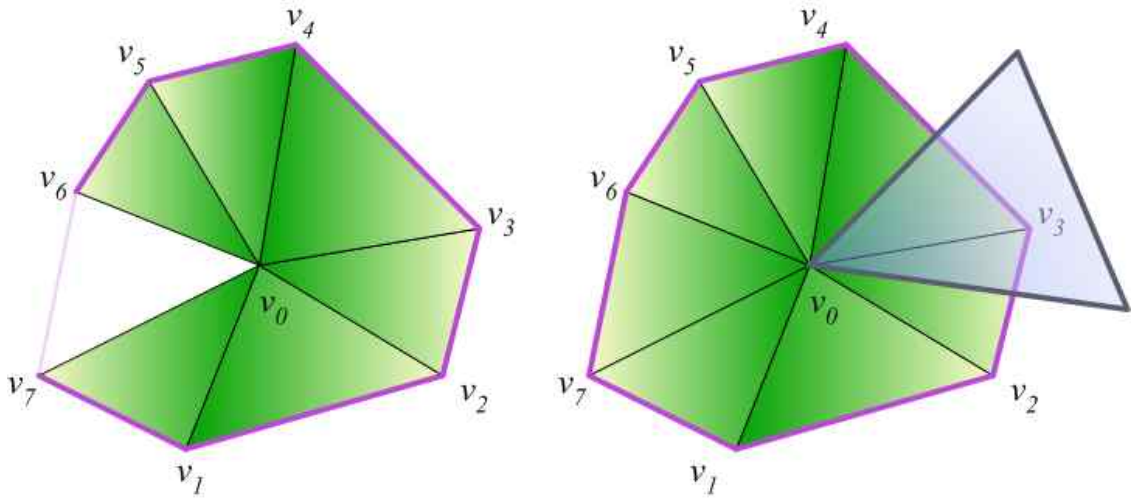

Abbildung 6.19: *Ein Keil entspricht der elementaren Situation einer Simplifizierung, weshalb eventuelle Sonderfälle berücksichtigt werden müssen. Während das linke Bild einen Rand beschreibt, droht im rechten Bild das Auseinanderreißen des Dreiecksnetzes in zwei Zusammenhangskomponenten. Beide Fälle können durch das Traversieren eines Zyklus identifiziert werden.*

weise die betreffenden Scheitelpunkte zwar nicht von der Simplifizierung ausgeschlossen, erhalten aber eine sehr niedrige Priorität. Stattdessen sind zunächst diejenigen Scheitelpunkte zu entfernen, die über einen kleinen Fehler verfügen und somit auch nur geringe Änderungen in Bezug auf das gesamte Dreiecksnetz verursachen. Abbildung 6.20 illustriert die Berechnung des Fehlers für einen Scheitelpunkt (siehe [SSB04a]). Von allen n Dreiecken eines Keils werden die Normalen kalkuliert und in einen gemeinsamen Punkt verschoben. Dort spannen die Normalen eine achsenparallele AABB mit dem Volumen V auf, dessen Diagonale D bestimmt wird. Der Fehler ϵ_v für einen Scheitelpunkt v ergibt sich dann aus folgender Formel:

$$\epsilon_v = \frac{V^2 \cdot \|D\|^2}{n} \tag{6.4}$$

Durch das Quadrat der Norm der Diagonalen D entfällt die Berechnung der Wurzel, weshalb ϵ_v schnell und einfach zu bestimmen ist. Entscheidende Faktoren sind im wahrsten Sinne des Wortes das Volumen und die Diagonale beziehungsweise das Verhältnis der beiden Werte zueinander. Die Länge einer Normalen spiegelt den Flächeninhalt des jeweiligen Dreiecks wieder [6]. Je größer der Flächeninhalt eines Dreiecks, desto länger seine Normale. Lange Normalen implizieren in der Regel eine größere AABB und damit einen größeren Fehler als kürzere Normalen. Dies bedeutet, dass aus einem Dreiecksnetz zunächst die unauffälligeren kleinen Dreiecke hinsichtlich der Simplifizierung berücksichtigt werden. Neben der Größe des Flächeninhalts hat aber auch die Krümmung des Keils beziehungsweise die Lage der Dreiecke zueinander einen enormen Einfluss. Liegen die Dreiecke alle in einer planaren Ebene, so

[6] Seien P_1, P_2 und P_3 die Punkte eines Dreiecks T. Interpretiert als Ortsvektoren ergeben die Punkte P_1, P_2 und P_3 mit $a = P_2 - P_1$ und $b = P3 - P_2$ die Basisvektoren a und b für die durch T aufgespannte Ebene. Das Vektorprodukt $a \times b$ liefert dann die Normale, deren Länge dem doppelten Flächeninhalt von T entspricht.

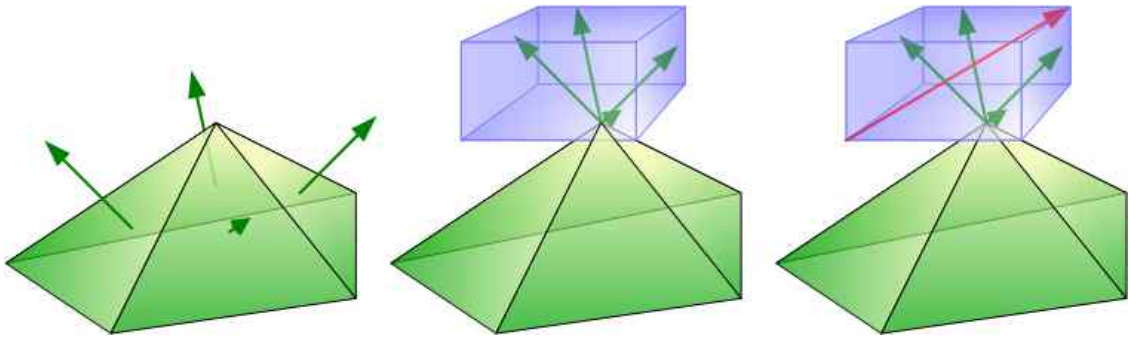

Abbildung 6.20: *Für die Berechnung der Metrik werden zunächst die Normalen der Dreiecke eines Keils bestimmt und imaginär in einen gemeinsamen Punkt verschoben. Dort spannen die Normalen eine achsenparallele Region auf, deren Diagonale bestimmt wird.*

gilt für die durch die Normalen aufgespannte AABB das Volumen $V = 0$. Entsprechend ist ein derartiger Keil der optimale Kandidat für die Simplifizierung, weil die Entfernung des korrespondierenden Scheitelpunkts zumindest bezüglich der Form des Modells keinen visuellen Effekt hat. Stark gekrümmte Bereiche repräsentieren dagegen in vielen Fällen die Charakteristik eines Modells. Sie werden aufgrund der größeren Streuung der Normalen mit einem größeren Volumen und folglich auch mit einem größeren Fehler bedacht. An dieser Stelle kommt zusätzlich die Diagonale zum Tragen. Das Volumen V kann relativ gering ausfallen, obgleich die Länge der Diagonalen D durchaus beträchtlich ist. Eine derartige Situation ist beispielsweise im Falle einer äußerst flachen AABB gegeben, da hier zwei der drei Seitenlängen der AABB große Werte beanspruchen, die dritte Seitenlänge aber sehr kurz ausfällt. Eine entsprechende AABB ist gleichbedeutend mit einer extremen Krümmung des Dreiecksnetzes in diesem Bereich und lässt auf einen Kantenbruch schließen. Aus diesem Grund erhöht die Diagonale den resultierenden Fehler. Im umgekehrten Fall entspricht die AABB nahezu einem Würfel und die im Vergleich zum Volumen kurze Diagonale betont die Eignung des betreffenden Keils für die Simplifizierung. Die Division durch die Anzahl der Dreiecke n ergibt mit wachsendem n einen sinkenden Fehler ϵ_v. Folglich werden vorzugsweise Keile mit einer hohen Zahl an Dreiecken simplifiziert, weil dort eine bessere Aussicht besteht, eine Vereinfachung mit lediglich geringen visuellen Konsequenzen zu erreichen.

Die Kalkulation der Fehler ist gleichbedeutend mit einer Priorisierung der Scheitelpunkte in Bezug auf den Half-Edge Collapse Operator. Insofern wird der Scheitelpunkt v mit dem kleinsten Fehler zuerst für die Simplifizierung selektiert, worauf innerhalb des zu v korrespondierenden Keils die Suche nach einer geeigneten Kante e für das Anwenden des Half-Edge Collapse Operators beginnt. Als Standardlösung kommt zunächst die kürzeste, v indizierende Kante des Keils in Frage, weil deren Entfernung mutmaßlich die geringste Auswirkung auf die Approximation an das Originalmesh m^0 bedeutet. Die Identifikation der kürzesten Kante kann dahingehend optimiert werden, dass selbige höchstwahrscheinlich Bestandteil des Dreiecks mit dem geringsten Flächeninhalt ist. Unter bestimmten Umständen stellt die kürzeste Kante hinsichtlich des Half-Edge Collapse Operators keine geeignete Auswahl dar. Hierzu zählen unter anderem Situationen, in denen die betreffende Kante eine Umkehrung mindestens eines Normalenvektors der durch den Half-Edge Collapse Operator veränderten Dreiecke verursachen würde. Beim Auftreten solcher Fälle ist daher die Suche auf die nächst kürzeste Kante auszudehnen. Nach dem Anwenden des Half-Edge Collapse Operators erfolgt

die Neuberechnung des Fehlers für die im Keil verbleibenden Scheitelpunkte. Anschließend wiederholt sich der Vorgang mit der Selektion desjenigen Scheitelpunkts, welcher nun über den kleinsten Fehler verfügt.

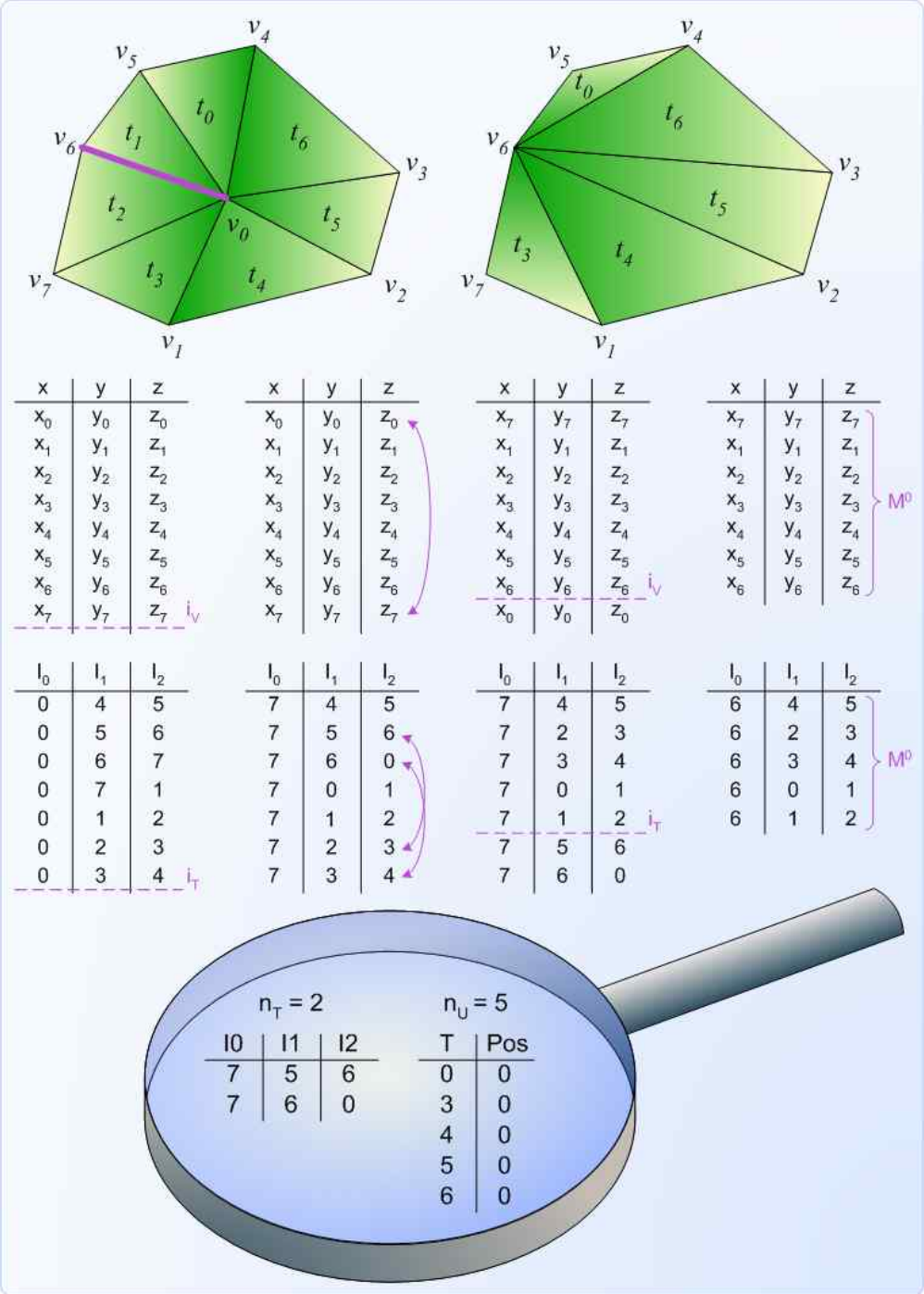

Abbildung 6.21: *Die Simplifizierung eines Keils und die daraus resultierenden progressiven Daten. Die dargestellten Tabellen der Scheitelpunkte beziehungsweise Dreiecke beinhalten keine Lücken und sind daher stets in einem für die schnelle Visualisierung geeigneten Format.*

Während der vorangehende Teil sich ausschließlich mit der Vereinfachung eines Dreiecksnetzes auseinandersetzt, ist nun die Frage, wie dabei ein progressives Format erzeugt werden kann, welches umgekehrt die schrittweise Rekonstruktion des Netzes ermöglicht. Abbildung 6.21 gibt hierzu einen Überblick, was nach der Auswahl einer Kante für die Anwendung des Half-Edge Collapse Operators geschieht. Kern der Beobachtung ist ein Keil, der durch eine Liste von acht Scheitelpunkten $V = \{v_0, \ldots, v_7\}$ und eine Liste von sieben Dreiecken $T = \{t_0, \ldots, t_6\}$ beschrieben ist. Für beide Listen existiert ein Index $i_V = 7$ beziehungsweise $i_T = 6$, welcher die letzte gültige Position innerhalb der betreffenden Liste adressiert. Ein Dreieck besteht aus drei Indizes, welche jeweils die Position des referenzierten Scheitelpunktes innerhalb von V wiedergeben. Voraussetzung ist, dass ausgehend von einem Scheitelpunkt die ihn indizierenden Dreiecke effizient ermittelt werden können.

Durch die Kantenkollabierung wird der Scheitelpunkt v_0 aus dem Dreiecksnetz entfernt. v_0 tauscht die Position mit dem durch i_V referenzierten Scheitelpunkt v_7 in V. Entsprechend müssen die Indizes zu den betroffenen Punkten in T angepasst werden, weshalb Index Null zu Index Sieben wechselt und umgekehrt. Anschließend wird i_V um eins reduziert. Die Dreiecke t_1 und t_2 fallen ebenfalls der Kantenkollabierung zum Opfer. Ihnen widerfährt ein ähnliches Schicksal wie v_0, d.h. sie tauschen nacheinander die Positionen mit den durch i_T indizierten Dreiecken von T, wobei i_T insgesamt um zwei reduziert wird. Der Trick ist, zwar i_V beziehungsweise i_T zu reduzieren, aber die hinter den beiden Indizes liegenden Einträge nicht zu löschen. Diese beinhalten nämlich Informationen, die später noch benötigt werden. Die verbleibenden Dreiecke t_0, t_3, t_4, t_5 sowie t_6 erhalten mit dem Zielpunkt v_6 der Kollabierung einen neuen Scheitelpunkt und müssen entsprechend verändert werden.

Nach der Simplifizierung beschreiben die Scheitelpunkte im Bereich vor i_V die Geometrie des Basismeshes M^0. Analoges gilt für die Dreiecke vor i_T innerhalb von T. Die Einträge hinter i_V beziehungsweise i_T repräsentieren exakt die Kantenkollabierungen in der inversen Reihenfolge ihrer Ausführung. Der vergrößerte Bereich unterhalb der Lupe in Abbildung 6.21 beinhaltet die progressiven Daten P, die für die Umkehrung jeweils einer Kantenkollabierung benötigt werden. Als Beispiel kann die Rekonstruktion des in Abbildung 6.21 simplifizierten Keils dienen. Zunächst muss das Basismesh mitsamt den sieben Scheitelpunkten und den fünf Dreiecken vorhanden sein, weshalb V und T die entsprechenden Werte beinhalten. Die progressiven Daten verfügen über eine Liste $U = \{0, 3, 4, 5, 6\}$, in welcher der Index aller nach der Kantenkollabierung verbleibenden Dreiecke t_0, t_3, t_4, t_5 sowie t_6 eingetragen ist, die mit dem Zielpunkt der Operation den neuen Scheitelpunkt v_6 erhielten. Dieser Zielpunkt ist nun wieder durch den Originalpunkt v_0 zu ersetzen. Glücklicherweise stehen die Einträge hinter i_V genau in der richtigen Reihenfolge, d.h. sie müssen einfach nur an V angefügt werden. Im konkreten Fall nimmt v_0 einen Platz am Ende von V ein, wodurch sich die Anzahl der Scheitelpunkte auf die ursprünglichen acht erhöht. Die Anzahl der Scheitelpunkte minus eins, also sieben, entspricht dem aktuellen Index des Originalpunktes v_0 vor der Kantenkollabierung. Aus diesem Grund werden die fünf durch U referenzierten Dreiecke mit dem betreffenden Index Sieben überschrieben. Hierfür ist zusätzlich die in U vermerkte Position des Originalpunktes v_0 innerhalb der Dreiecke t_0, t_3, t_4, t_5 sowie t_6 erforderlich. Insofern hat die Position einen Wertebereich von 0 bis 2. Im nächsten Schritt werden die durch die Kantenkollabierung entfernten Dreiecke t_1 und t_2 wieder rekonstruiert. Die entsprechenden Informationen zu t_1 und t_2 stehen in einer weiteren Liste $R = \{t_1, t_2\}$ der progressiven Daten

P und können an das Ende von T eingefügt werden [7]. Die im Beispiel gegebene Anzahl von zwei entfernten Dreiecken ist typisch für einen sauberen Keil. Erst mit der Behandlung von Sonderfällen können größere Zahlen auftreten. Der Vorgang der Verfeinerung ist mit dem Einfügen der Dreiecke t_1 und t_2 beendet. Ein Tauschen der Position wie im Falle der Simplifizierung ist weder in Bezug auf die Scheitelpunkte noch hinsichtlich der Dreiecke vonnöten, da die Reihenfolge der Scheitelpunkte beziehungsweise der Dreiecke bei der Visualisierung keine Rolle spielt [8].

Welche Konsequenz hat nun die oben beschriebene Vorgehensweise bei der Übertragung der visuellen Informationen eines Animationsagenten? Zunächst muss das Basismesh M^0 an das jeweilige Endgerät übertragen werden. Eine grundlegende Bedingung ist, dass jedes Endgerät in der Lage ist, dieses Basismesh in Echtzeit zu visualisieren. Dafür besteht allerdings eine sehr große Wahrscheinlichkeit, weil durch die Auflösung der Topologie eine hohe Kompression erreicht wird. Nach dem Transfer von M^0 ist zu entscheiden, wieviele Dreiecke ein Client erhalten soll. Die zu den Dreiecken korrespondierende Zahl an Scheitelpunkten kann beispielsweise über das in Abschnitt 4.1.1 erläuterte *Euler-Poincaré-Theorem* abgeschätzt werden. Angenommen, der Client soll nun zehn weitere Dreiecke empfangen. Jeder progressive Datensatz P beschreibt genau eine Inversion einer Kantenkollabierung, d.h. einen möglichen Schritt der Verfeinerung. Innerhalb von P gibt die Liste R die Anzahl der Dreiecke vor, die bei der Umkehrung der Kantenkollabierung eingefügt werden müssen. Üblicherweise sind das zwei Dreiecke, in extremen Fällen fünf bis sieben. Über die Größe von R kann die Zahl der zu übertragenden progressiven Datensätze bestimmt werden. Die exakte Menge von zehn Dreiecken wäre im optimalen Fall mit fünf Datensätzen abgedeckt. Es kann aber auch je nach Wahl zu einer geringfügigen Unter- beziehungsweise Überschreitung der Grenze von zehn Dreiecken kommen. Jeder transferierte progressive Datensatz repräsentiert aufgrund der Kantenkollabierung das Entfernen eines Scheitelpunkts. Als Konsequenz muss neben dem progressiven Datensatz der korrespondierende Scheitelpunkt übertragen werden. Die entsprechenden Scheitelpunkte stehen nach der Simplifizierung hinter dem Index i_V genau in der richtigen Reihenfolge, weshalb im optimalen Fall von fünf progressiven Datensätzen lediglich der Transfer der fünf auf i_V folgenden Punkte in V erforderlich ist. Nach ihrer Ankunft auf Seite des Clients können die Scheitelpunkte einfach an das Ende der dort bereits existierenden Punktliste angefügt werden. Eine Veränderung ist nur hinsichtlich der durch die progressiven Datensätze betroffenen Dreiecke notwendig.

Nach Abschnitt 5.1.1 ist eine weitere Anforderung die Berücksichtigung möglicher Zusatzattribute. Um welche Attribute es sich im konkreten Fall handelt, ist durch die Dependency beschrieben. Innerhalb einer Dependency ist zwischen per Vertex und per Facet Informationen zu unterscheiden. Erstere werden analog zu den Scheitelpunkten, letztere analog zu den Dreiecken vertauscht. Da ein Vertauschen während der Simplifizierung bei einer großen Anzahl von Zusatzattributen zum Verschieben enormer Datenmengen führen kann, prozessiert die Simplifizierung zunächst ausschließlich die notwendigerweise in der Dependency vor-

[7] In Bezug auf R ist es wichtig, die Dreiecke t_1 und t_2 während der Simplifizierung erst nach dem Vertauschen von v_0 und v_7 in R einzutragen. Aus diesem Grund beinhalten die entsprechenden Dreiecke beim Rekonstruieren den korrekten Index zu v_0, sobald selbiger an das Ende von V angefügt wird.

[8] Eine Ausnahme wären betrachterabhängige Verfeinerungen, wie sie etwa von Hoppe et al. [Hop97] vorgeschlagen werden.

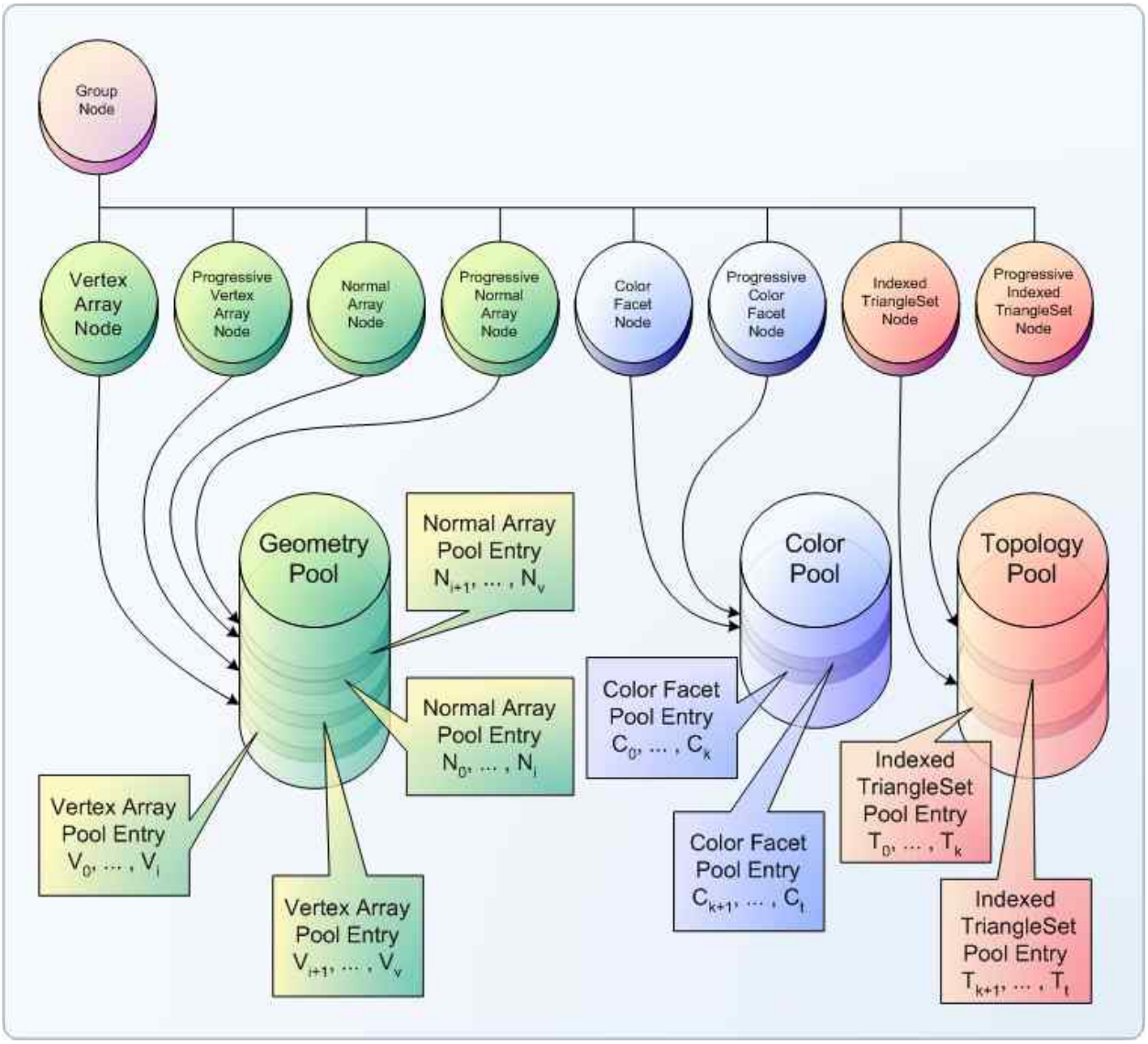

Abbildung 6.22: *Nach der Simplifizierung erhält jeder Knoten des Elementgraphen, der eine Information pro Scheitelpunkt oder pro Fläche spezifiziert, einen progressiven Partnerknoten. Letzterer verweist auf die progressiven Daten, welche auf die Informationen des Basismeshes folgen.*

handenen Scheitelpunkte und Dreiecke. Eine Dependency muss also unbedingt neben dem Topologieknoten mindestens einen korrespondierenden Vertex Array Node beinhalten. Nach der Simplifizierung werden dann die Zusatzattribute anhand der durchgeführten Operationen in die richtige Reihenfolge sortiert. Angenommen im Falle des obigen Beispiels von zehn zusätzlich übertragenen Dreiecken sollen nun auch noch die passenden per Vertex Normalen und per Facet Farben übertragen werden. Für das Basismesh M^0 entspricht die Anzahl der Normalen der Menge der Scheitelpunkte und die Anzahl der Farben der Menge der Dreiecke. Gleiches gilt ebenfalls bezüglich der nachfolgenden Daten. Wie die Scheitelpunkte stehen auch die Normalen hinter dem Index i_V in der passenden Reihenfolge. Dasselbe trifft auf die Farben hinter dem Index i_T zu. Beide Datensätze müssen daher nach ihrer Ankunft auf Seite des Clients lediglich an die jeweiligen Daten des Basismeshes angefügt werden.

Abbildung 6.22 illustriert wie die Daten nach der Simplifizierung eines Elementgraphen auf Seite des Servers verwaltet werden. Die ursprünglich durch den Graphen referenzierten Pooleinträge beschreiben nun ausschließlich das Basismesh M^0. Sämtliche per Vertex Daten hinter i_V sowie per Facet Daten hinter i_T werden einschließlich der Scheitelpunkte und der Dreiecke in jeweils einem gesonderten Pooleintrag abgelegt. Jeder Knoten des Elementgraphen mit Ausnahme der Transform Nodes und der Group Nodes erhält einen progressiven Partnerknoten, der direkt hinter dem jeweiligen Knoten eingetragen wird. Der erste Knoten adressiert also immer die zugehörigen Daten des Basismeshes und der folgende progressive Knoten die restlichen Informationen in der zur Kantenkollabierung inversen Reihenfolge.

Sobald ein Animationsagent an einen Client zu übertragen ist, können mit Hilfe des im Agenten vermerkten Elementgraphen die visuellen Informationen schnell identifiziert werden. Gleiches gilt für das an erster Stelle zu übermittelnde Basismesh. Alle darüber hinausgehenden Daten können nahezu exakt an die vorgegebene Menge von Dreiecken angepasst werden. Die Fehlertoleranz sollte sich selbst in extremen Fällen auf maximal fünf bis sieben Dreiecke beschränken. Serverseitig stehen die Daten für die Übertragung bereits in der richtigen Sequenz und müssen lediglich am Stück ausgelesen und kodiert werden. Bei den Scheitelpunkten sowie allen Zusatzattributen sind für das Einfügen auf Seite des Clients keine weiteren Informationen erforderlich, weshalb die jeweiligen Daten 1:1 als getrennte Datenströme übermittelt werden können. Hierdurch ist die in Abschnitt 5.1.1 verlangte Auftrennung der Datenströme gegeben. Lediglich in Bezug auf die Dreiecke sind in Form der progressiven Datensätze weitere Informationen vonnöten. Die entsprechenden Daten fallen aber äußerst gering aus. Das Einfügen selbst basiert auf einigen wenigen, kostengünstigen Operationen: Scheitelpunkte und Zusatzattribute werden einfach an die existierenden Daten angefügt, Dreiecke erfordern zusätzlich eine gesonderte Behandlung einiger weniger Indizes. Die entsprechenden Operationen liegen daher auch für extrem schwache Endgeräte im Bereich des Möglichen. Durch die Sortierung der Daten existiert die Option zur paketweisen Übertragung und Verfeinerung eines Modells.

Ein weitere positive Eigenschaft des vorgestellten Konzepts zur Erzeugung progressiver Datenströme wird durch das Beispiel in Abbildung 6.21 deutlich. Sowohl die Liste der Scheitelpunkte V als auch die Liste der Dreiecke T beschreiben zusammen mit den Indizes i_V und i_T stets eine kompakte, geschlossene Darstellung eines Meshes, d.h. es treten von Position zu Position keine Lücken auf. Selbiges gilt auch für alle Zusatzattribute. Aus diesem Grund ist das Modell während der Simplifizierung sowie der Verfeinerung immer in einem Zustand, welcher das effiziente Rendern des Modells etwa für eine Visualisierung erlaubt (siehe [BSS03]). Insofern ist im Gegensatz zu anderen Verfahren keine teure Traversierung mehrerer verschachtelter Datenstrukturen erforderlich, um die entsprechenden Daten zu sammeln. Damit sind alle Anforderungen aus Abschnitt 5.1.1 erfüllt.

6.5 Die Scheduler-Komponente

Ein entscheidendes Problem hinsichtlich der Adaption der Daten ist die geeignete Auswahl der Menge an Elementen, die zu einem Client übertragen werden müssen. Abschnitt 5.1.5

geht hierfür auf den Begriff der Area-of-Interest eines Clients ein, wonach der Server einem Client nur diejenigen Elemente einer Szene übermittelt, die sich innerhalb der Area-of-Interest des betreffenden Clients befinden. Um auf Seite des Servers die Area-of-Interest identifizieren zu können, muss dort jeder Client gemäß dem Area-of-Interest Konzept in einer bestimmten Form repräsentiert sein. Im Falle eines visuellen Konzepts handelt es sich dabei oftmals um den Sichtbereich des Betrachters, d.h. der Client überträgt in regelmäßigen Abständen die aktuellen Sichtparameter seines Benutzers an den Server. Mit dem in Abschnitt 5.2.2 beschriebenen Client-Server-System RING liefert Funkhouser [Fun95] den Beweis, dass mit Hilfe visueller Kriterien die Kosten des Nachrichtenaustauschs hinsichtlich der Synchronisation dynamischer 3D Szenen drastisch reduziert werden können. Nach Funkhousers Intention ist auf einem Client lediglich die Aktualisierung derjenigen transformierten Elemente einer Szene erforderlich, welche im Sichtbereich des betreffenden Benutzers liegen. Hieraus lassen sich zwei Schlussfolgerungen ableiten:

- Der Server muss an einen Client nur Nachrichten versenden, welche die Aktualisierung dort sichtbarer Elemente betreffen.
- Der Server muss wissen, welche Elemente der Szene in Bezug auf einen Client sichtbar sind.

Funkhousers Idee kann auf den Transfer der visuellen Informationen einer Szene erweitert werden. In diesem Fall ist nur die Übertragung der auf einem Client sichtbaren Animationsagenten vonnöten. Gleiches gilt für die zum Agenten gehörenden Informationen wie etwa der Elementgraph oder die Pooleinträge. Die schrittweise Verfeinerung eines Modells aufgrund progressiver Datenpakete ist in der Aktualisierung des Modells mit einbegriffen.

Die Übertragung der visuellen Informationen bildet zusammen mit der zweiten oben aufgeführten Schlussfolgerung den Ausgangspunkt für ein gewaltiges Malheur: Wie kann der Server in Echtzeit entscheiden, welche Elemente einer großen Szene auf eventuell mehreren hundert Clients sichtbar sind? Erschwerende Faktoren stellen die Navigation der Benutzer sowie die Dynamik der aktiven Elemente dar. Von Seiten der Clients darf der Server nicht auf Unterstützung hoffen. Damit die Clients selbst entscheiden könnten, welche Elemente für sie sichtbar und dementsprechend vom Server anzufordern sind, müssten sie die betroffenen Regionen der Szene kennen. Unglücklicherweise sollen aber gerade die dort vorhanden Elemente übertragen beziehungsweise aktualisiert werden. Ein Client müsste somit ein Element als sichtbar deklarieren, welches er gar nicht kennen dürfte, weil es ihm ja erst übermittelt werden soll. An dieser Stelle existiert also ein Teufelskreis, dessen einziger Ausweg darin besteht, dass der Server mit einer geeigneten Lösung aufwarten kann.

Im speziellen Fall von Funkhousers Arbeit handelt es sich hierbei um ein PVS gestütztes Occlusion Culling Verfahren (siehe Abschnitt 4.6). Funkhouser verwendet als Beleg seines Konzepts ein Innenraummodell, welches er in mehrere Zellen unterteilt und entsprechend auf dem Server repräsentiert. Besteht eine Szene aus k Zellen, so werden für jede Zelle Z_i mit $i = 1, \ldots, k$ während eines Vorverarbeitungsschrittes die von Z_i aus potentiell sichtbaren Nachbarzellen bestimmt und dort konserviert. Sobald die Repräsentation eines Clients A, also dessen Avatar, auf Seite des Servers eine Zelle Z_i betritt, erfolgt anhand der in der Zelle

gespeicherten Daten die schnelle Identifizierung der betreffenden Nachbarzellen. Die dort registrierten Clients $C_1, \ldots, C_n$ müssen nun hinsichtlich A aktualisiert und umgekehrt A auf $C_1, \ldots, C_n$ hingewiesen werden.

Innenarchitekturen bieten für gewöhnlich den Vorteil großer Verdeckungstiefen, weshalb die Sichtbereiche der Betrachter enormen Einschränkungen unterliegen. PVS basierte Ansätze sind prädestiniert für derartige Szenarien, zum einen weil bis auf die Benutzer selbst keine dynamischen Elemente in den Szenen auftreten und zum anderen weil die pro Zelle zu speichernden Informationen sich in einem engen Rahmen halten. Leider ist eine Erweiterung auf generische 3D Szenen kaum möglich. PVS Verfahren verlangen zwar während der Laufzeit nur einen sehr geringen Aufwand für das Abrufen der Informationen, jedoch bedürfen sie einer äußerst teuren Vorberechnung. Dynamische Vorgänge sind daher nur unter Inkaufnahme neuer zeitintensiver Kalkulationen möglich. Zudem beanspruchen Szenen mit geringerer Verdeckungstiefe einen ungleich höheren Speicherraum.

Andere Occlusion Culling Verfahren unterstützen analog dem PVS Ansatz entweder keine dynamischen Szenen oder bauen auf die Unterstützung von Seiten der Hardware [BMH99, ZMHHI97]. Zwar ist unter Einbeziehung der Hardware eine effiziente Lösung des Sichtbarkeitsproblems für generische Szenen im Bereich des Möglichen, allerdings trifft dies nur in Bezug auf einen einzelnen Betrachter zu. Ein Standard PC ist mit Sicherheit nicht in der Lage, für mehrere Clients ein Occlusion Culling in Echtzeit durchzuführen. An dieser Stelle bliebe dann nur die Alternative, als Server einen grafikfähigen Superrechner zu verwenden, wie er etwa von SGI angeboten wird.

Aus diesem Grund scheiden Occlusion Culling und View Frustum Culling Verfahren für die Bestimmung der visuellen Area-of-Interest eines Clients aus. Anstatt wie die eben genannten Techniken die genaue Sichtpyramide des Betrachters zu evaluieren, vereinfachen Hesina et al. [HS98] das Problem auf eine kreisförmige Region um die Position des Betrachters. Hierdurch benötigt der Server für die Selektion der zu übertragenden Elemente zwar deutlich weniger Rechenzeit als im Falle eines Occlusion Culling Verfahrens, allerdings ist diese Vorgehensweise auch wesentlich unpräziser. Deshalb steht zu erwarten, dass der Server als Konsequenz wesentlich mehr Elemente übertragen muss. Entsprechendes gilt für den Empfang der Daten auf Seite des Clients. Leider machen Hesina et al. keine genauen Angaben, wie sie eine 3D Szene repräsentieren und wie sie die wichtigen Elemente in Bezug auf mehrere Clients identifizieren. Insofern muss im Rahmen der vorliegenden Arbeit ein neues Area-of-Interest Konzept ausgearbeitet werden. Die Selektion der zu übertragenden Elemente anhand dieses Konzepts obliegt der Scheduler-Komponente.

6.5.1 Ein Area-of-Interest Konzept

Nach Abbildung 6.23 wird ein Client auf Seite des Servers durch drei ineinander verschachtelte AABBs repräsentiert. Die rote AABB im Innersten umschließt die Sichtpyramide des Betrachters und wird im folgenden als *sichtbare Zone* bezeichnet [9]. Elemente beziehungswei-

[9] An dieser Stelle ist anzumerken, dass die Sichtpyramide nicht wirklich auf dem Server existiert, sondern in Abbildung 6.23 nur die Lage der drei AABBs verdeutlichen soll. Die vom Client an den Server übermittelten Sichtparameter beinhalten somit ausschließlich die Daten der AABBs.

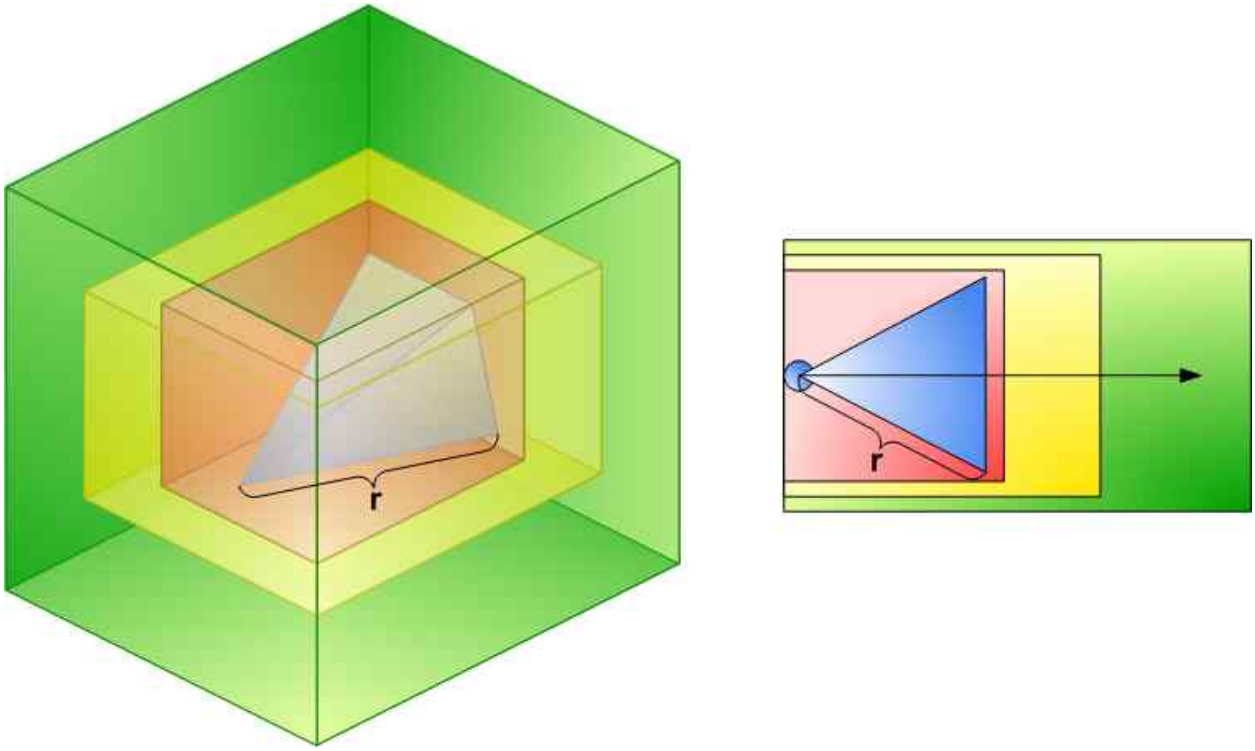

Abbildung 6.23: *Die Area-of-Interest eines Clients wird repräsentiert durch drei ineinander verschachtelte AABBs. Je näher eine Zone der Sichtpyramide liegt, desto höher ist ihre Dringlichkeit. Die Distanz r ergibt sich aus dem Abstand zwischen dem Augpunkt und einem der Scheitelpunkte auf der hinteren Clippingebene. Dieser Wert wird später für die in Abschnitt 6.7.1 beschriebene Deformation der Zonen benötigt.*

se Animationsagenten in dieser Zone erhalten in Bezug auf einen Client die höchste Priorität und müssen augenblicklich übertragen werden. Die mittlere, gelbe AABB berücksichtigt die Transformation der Sichtpyramide durch die Erfassung derjenigen Animationsagenten, welche infolge der Navigation des Benutzers in die sichtbare Zone eindringen könnten. Die betreffende AABB erhält daher die Bezeichnung *dynamische Zone*. Passend zur Lage dieser Zone im Vergleich zu den übrigen AABBs verfügen Animationsagenten innerhalb der dynamischen Zone über eine mittlere Priorität. Die äußerste, grüne AABB deckt die Bereiche einer Szene ab, welche in Bezug auf die Übertragung momentan zwar nicht kritisch sind, es in naher Zukunft aber werden könnten. Der folgende Text referenziert diese AABB als *präventive Zone*. Die Intention der präventiven Zone beruht darauf, Animationsagenten in den potentiell kritischen Regionen vorsorglich zu transferieren, sofern der Server mit der Prozessierung des Inhalts der beiden anderen AABBs nicht ausgelastet ist. Entsprechend werden Animationsagenten innerhalb der präventiven Zone mit der niedrigsten Priorität versehen.

Mit Hilfe der drei Zonen ist eine Priorisierung der Animationsagenten unter Berücksichtigung der Distanz zum Betrachterstandort gegeben. Unglücklicherweise stellen die achsenparallelen Zonen keine gute Approximation an die imaginäre Sichtpyramide dar. Liegt die Betrachterposition beispielsweise im Zentrum der sichtbaren Zone, so erhalten möglicherweise selbst Animationsagenten hinter der Projektionsebene des Betrachters die höchste Priorität. Aus diesem Grund erfolgt eine Adaption der Zonen an die zu erwartenden Bewegungen des Betrachters. Hierzu erfasst die Präsentation-Komponente auf Seite des Clients in bestimmten

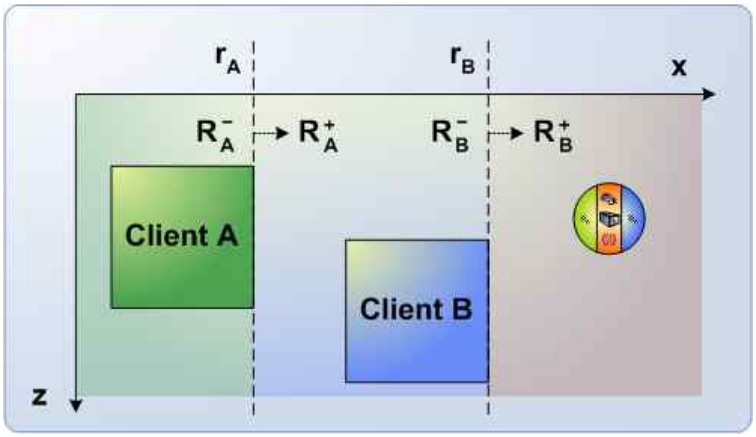

Abbildung 6.24: *Die Idee der Clientbäume ist die Minimierung der Vergleiche zwischen den Zonen und der Animationsagenten. Hierzu werden nicht nur die räumlichen Kohärenzen des Raumunterteilungsbaumes ausgenutzt, sondern auch die der Zonen selbst.*

Zeitabständen die Lage und die Position der Sichtpyramide des Benutzers. Anschließend ermittelt die Präsentation-Komponente anhand der gewonnenen Daten für jede der drei AABBs die erforderlichen Dimensionen und teilt diese dem Server mit. Auf diese Weise ist in Form der drei AABBs zwar ein gewisser Übertragungsaufwand zu leisten, andererseits wird der Server aber nicht mit der Erfassung der Benutzerdaten und deren Auswertung belastet. Die Prekalkulation der zu erwartenden Navigation des Benutzers sowie die daraus resultierende Anpassung der Area-of-Interest ist als Teil der Präsentation-Komponente im entsprechenden Abschnitt 6.7.1 erläutert. Aus Sicht der Scheduler-Komponente ist lediglich die grundlegende Beschreibung der Area-of-Interest durch drei ineinander verschachtelte AABBs und deren Priorität von Interesse. Sofern eine Zone mit höherer Priorität die Zonen mit niedrigerer Priorität stets vollständig umschließt, spielen die Größenverhältnisse der einzelnen Zonen an dieser Stelle keine Rolle. Nichtsdestotrotz vermittelt die rechte Seite von Abbildung 6.23 schon einmal einen kleinen Eindruck von der Adaption der Area-of-Interest an die Bewegung des Betrachters.

6.5.2 Die Konstruktion des Clientbaumes

Um einem Animationsagenten in Bezug auf einen Client eine Priorität zuordnen zu können, muss die räumliche Lage des Agenten mit den Zonen des Clients verglichen werden. Ein Agent erhält die Priorität einer Zone, sofern er sich entweder ganz oder zumindest teilweise im Inneren der betreffenden Zone befindet. Liegt ein Agent in mehreren Zonen, so ist die Zone mit der höchsten Priorität ausschlaggebend.

Prinzipiell ist es denkbar, sämtliche Animationsagenten einer Szene einzeln gegen die Zonen der Clients zu testen. Der damit verbundene Aufwand $O(n \cdot k)$ mit n gleich der Anzahl der Animationsagenten und k gleich der Anzahl der Zonen zeugt allerdings von keiner besonders effizienten Vorgehensweise. Selbst das hierarchische Culling gegen jeweils eine der k Zonen mit Hilfe des in Abschnitt 6.3.4 beschriebenen Raumunterteilungsbaumes bietet keine optimale Lösung, da die räumlichen Kohärenzen der Zonen untereinander nicht ausgenutzt

werden. Aus diesem Grund erfolgt über die Zonen der Clients die Konstruktion einer hierarchischen Baumstruktur, dem sogenannten *Clientbaum*. Das Konzept des Clientbaumes ist ähnlich zum Ansatz von Chin et al. [CF89], deren Intention in der Berechnung von Schattenwürfen liegt. Bittner et al. [BHS98] optimieren den Vorschlag von Chin et al. für die Repräsentation von Occludern im Rahmen eines Occlusion Culling Verfahrens.

Alle Abbildungen hinsichtlich des Clientbaumes illustrieren ausschließlich den zweidimensionalen Fall, weshalb die Ränder einer Zone hier Kanten entsprechen. Die Bezeichnungen l, u, r, o der Kanten stehen für *links*, *unten*, *rechts* beziehungsweise *oben* und indizieren immer die entsprechende Lage eines Randes. Sie werden in der angegebenen Sequenz behandelt. Clients werden mit Großbuchstaben referenziert und ihre Zonen stets nach der alphabetischen Reihenfolge der korrespondierenden Clients bearbeitet.

Abbildung 6.24 verdeutlicht in Bezug auf die Scheduler-Komponente die grundlegende Idee des Clientbaumes: Jede Fläche eines Zonenrandes spannt eine Hyperebene h auf, welche analog einem k-D-Tree den Raum in einen negativen und einen positiven offenen Halbraum h^- beziehungsweise h^+ unterteilt (siehe Abschnitt 4.4). Im Zusammenhang mit dem Clientbaum entspricht h^- dem inneren offenen Halbraum und h^+ dem äußeren offenen Halbraum. Wohingegen der äußere Halbraum vollkommen disjunkt zu der mit h assoziierten Zone ist, beinhaltet der innere Halbraum die komplette Zone. Die Normale der Hyperebene h weist in den äußeren Halbraum. Abbildung 6.24 zeigt zwei Zonen, die auf Seite des Servers einen Client A beziehungsweise einen Client B repräsentieren. Der Typ der Zonen ist für das Prinzip des Clientbaumes unerheblich. Der rechte Rand jeder Zone spannt jeweils eine Hyperebene r_A beziehungsweise r_B auf, welche den Raum in vier offene Halbräume R_A^+, R_A^-, R_B^+ sowie R_B^- unterteilt. Liegt nun ein Animationsagent im Halbraum R_B^+ und die Zone von A vollständig im Halbraum R_B^-, so impliziert dies automatisch, dass sich der Agent ebenfalls in R_A^+ befindet. Ein expliziter Test zwischen dem Animationsagenten und A ist also nicht mehr erforderlich.

Im folgenden wird nun die Konstruktion des Clientbaumes aus den Zonen der Clients beschrieben. Die Erläuterungen gehen davon aus, dass pro Area-of-Interest lediglich eine Zone beziehungsweise die mit der Zone assoziierte AABB in den Clientbaum eingefügt wird. Daher ist unter dem Einfügen eines Clients X in den Clientbaum die Bearbeitung der betreffenden AABB zu verstehen. Die Ursache für die Reduktion auf eine einzelne AABB je Client ist später in Abschnitt 6.5.4 erklärt.

Abbildung 6.25 illustriert einen Clientbaum nach dem Einfügen der Clients A und B. Der Baum besteht aus zwei verschiedenen Knotentypen, nämlich den *Clientknoten* und den *Zonenknoten*. Letztere spezifizieren jeweils eine Hyperebene, die durch die Fläche einer Zone aufgespannt wird. Zonenknoten entsprechen stets inneren Knoten des Clientbaumes und haben maximal zwei Kindknoten. In den Abbildungen liegt der linke Kindknoten eines Zonenknotens im inneren offenen Halbraum und der rechte Kindknoten im äußeren offenen Halbraum der Hyperebene. Die Blätter des Clientbaumes sind immer Clientnodes, welche Identifikationen bestimmter Clients beinhalten, d.h. die Scheduler-Komponente kann mit Hilfe eines Clientnodes sehr schnell die betreffenden Clients ermitteln.

Der Aufbau des Clientbaumes beginnt mit der Behandlung des Clients A. Als Wurzel dient ein Zonenknoten mit der Hyperebene l_A. Per Definition sind sämtliche Flächen der AABB

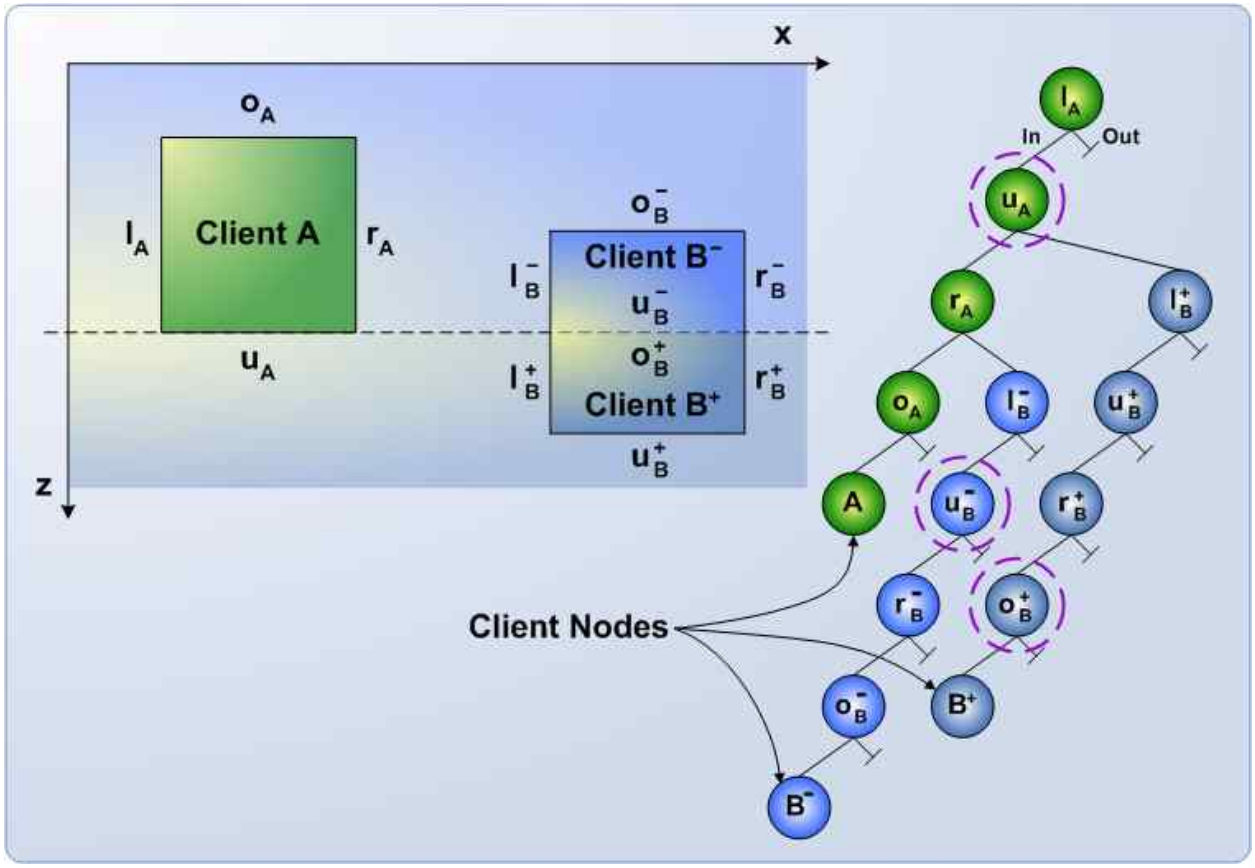

Abbildung 6.25: *Der Clientbaum nach dem Einfügen der Clients A und B. Die markierten Knoten beschreiben dieselbe Hyperebene, weshalb die beiden unteren Knoten überflüssig sind.*

einer Zone stets vollständig im inneren Halbraum einer Hyperebene, die durch eine Fläche der gleichen AABB aufgespannt wird.

Definition (Inklusion): Die Flächen einer AABB befinden sich stets im offenen inneren Halbraum des durch eine Hyperebene h unterteilten Raumes, sofern h über eine Fläche der gleichen AABB aufgespannt wird.

Aufgrund dieser Definition werden die Hyperebenen u_A, r_A und o_A ohne weiteren Test in Form von Zonenknoten gleich einer Perlenkette aufgereiht. Dabei ist zu beachten, dass die Zonenknoten u_A, r_A und o_A jeweils als linke Kindknoten eingetragen sind. Das Ende einer derartigen Perlenkette ziert immer ein Clientnode, welcher den mit der AABB assoziierten Client identifiziert. Für das Einfügen des Clients B wird der bereits existierende Clientbaum traversiert. B befindet sich vollständig im inneren Halbraum von l_A, weshalb eine Propagierung von B zu u_A erfolgt. Unglücklicherweise schneidet B die Hyperebene u_A, weshalb B in eine in Bezug auf u_A innere und äußere AABB unterteilt werden muss. Da u_A keinen rechten Kindknoten aufweist, wird die äußere AABB B^+ analog zu A ohne Test als rechter Teilbaum von u_A eingetragen. Die innere AABB B^- muss dagegen zunächst den Umweg über den linken Kindknoten r_A von u_A nehmen. Hier befindet sich B^- komplett im inneren Halbraum, weshalb B^- wie B^+ zuvor als rechter Teilbaum von r_A eingetragen wird. Aufgrund ihrer achsenparallelen Eigenschaft ist das Schneiden zweier AABBs einfach zu überprüfen.

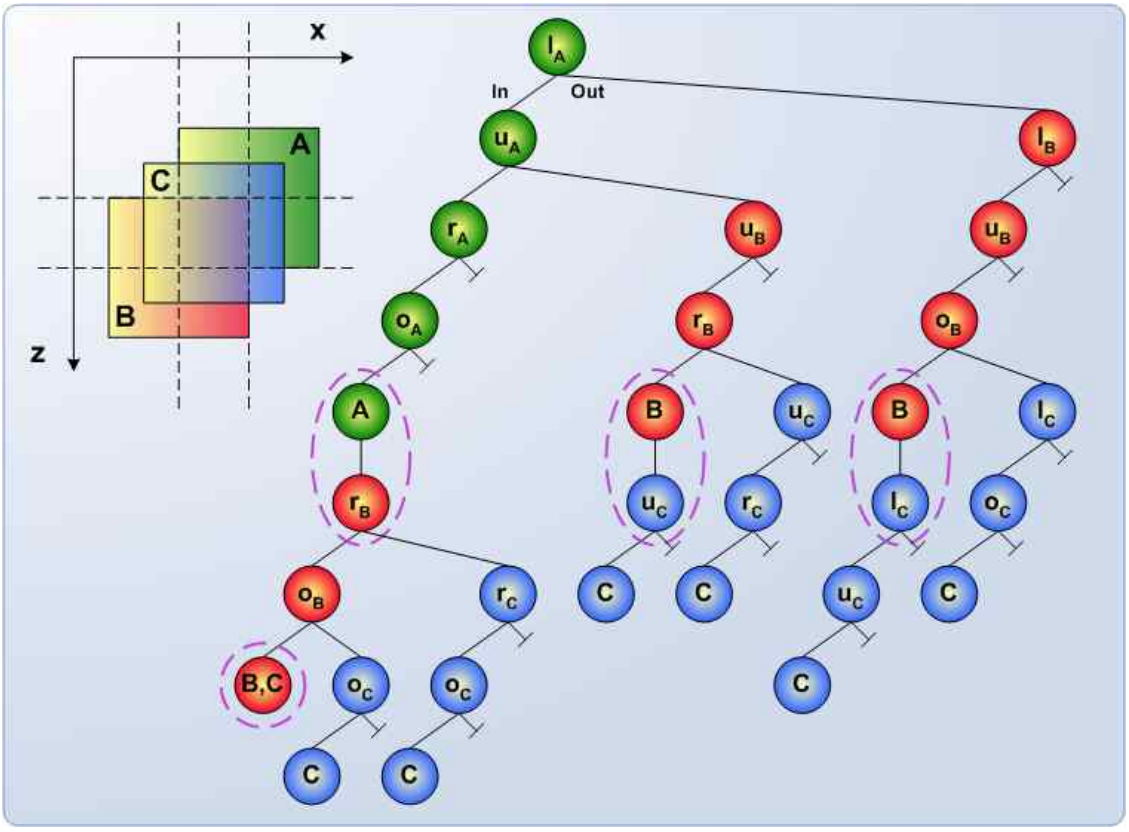

Abbildung 6.26: *Sobald sich die Areas-of-Interest gegenseitig überlappen, können Sonderfälle eintreten. In der ersten Situation erreicht eine AABB einen Clientknoten, ohne dass alle ihre Flächen markiert sind. Dadurch entstehen die mit Ellipsen markierten Verbindungen. Sind dagegen bereits alle Flächen markiert, so erhält der betretene Clientknoten eine weitere Clientidentifikation. Diese Situation ist mit einem Kreis umrandet.*

Gleiches gilt für das eventuell erforderliche Unterteilen einer AABB.

Jeder Pfad von der Wurzel bis zu einem Clientknoten beschreibt eine ganz bestimmte AABB. Aus diesem Grund ist es unnötig, innerhalb eines Pfades mehrere Zonenknoten zu halten, welche die gleiche Hyperebene repräsentieren. Abbildung 6.25 kennzeichnet derartige Knoten mit einem gestrichelten, roten Kreis. Demnach liegen u_A und u_B^- sowie u_A und o_B^+ im selben Pfad, obgleich sie eine äquivalente Hyperebene beschreiben. Der Ursprung von u_B^- und o_B^+ liegt in der Unterteilung von B in Bezug auf u_A begründet. Daher schlagen Bittner et al. vor, sämtliche aus einer Unterteilung resultierenden Ränder einer AABB zu markieren. Beim Einfügen einer unterteilten AABB in den Clientbaum werden dann bereits markierte Flächen einfach nicht mehr berücksichtigt, wodurch sich die Länge der Perlenketten mit wachsender Komplexität des Clientbaumes deutlich reduziert.

Im Gegensatz zu den Ansätzen von Chin et al. und Bittner et al. ist es im Falle der Scheduler-Komponente nicht ausreichend, den Clientbaum als einen vollständig binären Baum anzusehen. Abbildung 6.26 illustriert hierzu ein Szenario mit drei sich gegenseitig überlappenden Clients A, B und C. Da sich ein Teil von B im inneren von A befindet, erreicht beim Einfügen von B zumindest ein Teilbereich B' den Clientknoten A. Dort werden alle nicht markierten Flächen beziehungsweise Hyperebenen von B' als einzelner Teilbaum an A angehängt.

Allgemein formuliert: Wird ein Client Y in einen Clientbaum T eingefügt und schneidet Y einen bereits in T vorhandenen Client X, so beschreiben die an die Clientknoten von X angehängten Teilbäume die Schnittregionen von X und Y.

Ein Verwerfen der Informationen von B' ist ausgeschlossen, da ein Animationsagent zwar durchaus in A enthalten sein kann, in Bezug auf B aber nicht zwangsläufig das gleiche gelten muss. Als Konsequenz sind Clientknoten nicht automatisch Blätter des Clientbaumes. Abbildung 6.26 kennzeichnet die entsprechenden Fälle mit einer gestrichelten Ellipse. Unter Umständen erreicht eine AABB einen Clientknoten und sämtliche Ränder der AABB sind durch Unterteilungen markiert. Eine derartige Situation tritt auf, wenn wie in Abbildung 6.26 ein Client C eingefügt wird, der von den im Clientbaum schon existierenden Clients A und B vollständig umschlossen ist. Hier ist es weder möglich noch notwendig einen Teilbaum an den Clientknoten anzuhängen. Schließlich beschreibt der Pfad bis zu dem Clientknoten bereits die AABB. Aus diesem Grund wird der mit der AABB assoziierte Client zusätzlich in den Clientknoten eingetragen. Ein Clientknoten kann also durchaus mehrere Identifikationen beinhalten. Abbildung 6.26 indiziert den Knoten durch einen gestrichelten Kreis.

Die Effizienz des Clientbaumes ist bedingt durch die Lage der nacheinander eingefügten Zonen. Im ungünstigsten Fall bilden diese eine diagonale Linie und werden ausgehend von einer der beiden Endzonen entweder auf- oder absteigend dem Clientbaum hinzugefügt. Ein derartiges Szenario entspricht dem Erstellen eines binären Suchbaumes anhand einer bereits sortierten Sequenz von Zahlenwerten: Basiert die Wurzel des Binärbaumes auf der ersten und damit auf der kleinsten beziehungsweise größten Zahl, so werden alle folgenden Werte nur einem Teilbaum zugeordnet. Hierdurch entsteht ein extrem unbalancierter Baum. Um dieses Problem im Falle des Clientbaumes zu vermeiden, kann eine alternierende Vorsortierung der Zonen entlang den Hauptachsen erfolgen. Durch die in der Regel beschränkte Zahl an Clients hält sich der Aufwand in einem realistischen Rahmen. In der Praxis erzielen aber zufällig gewählte Reihenfolgen im Durchschnitt keine wesentlich schlechteren Ergebnisse.

6.5.3 Die effiziente Auswertung des Clientbaumes

Da sich die Zonen durch die Navigation der Clients verschieben können, aktualisiert die Scheduler-Komponente in regelmäßigen Zeitabständen den Clientbaum. Sobald der Aufbau des Clientbaumes abgeschlossen ist, testet die Scheduler-Komponente die Szene gegen den Clientbaum. Eine grundlegende Bedingung ist dabei die Repräsentation der Szene durch eine hierarchische Struktur wie etwa einem Raumunterteilungsbaum oder eine Bounding Volume Hierarchie. Es muss sich also nicht zwangsläufig um den in Abschnitt 6.3.4 erläuterten Raumunterteilungsbaum handeln. Als Einschränkung gilt jedoch, dass die Zelle eines übergeordneten Knotens die Zellen aller Kindknoten vollständig umschließen muss. Weiterhin sind lediglich achsenparallele Konzepte wie beispielsweise ein Octree oder ein k-D-Tree erlaubt. Falls der in Abschnitt 6.3.4 beschriebene Raumunterteilungsbaum verwendet wird, ist zu beachten, dass Animationsagenten sowohl einzeln durch einen Agentknoten als auch gruppiert durch einen Hierarchieknoten repräsentiert werden können. Erhält ein Hierarchieknoten eine bestimmte Priorität, so ist selbige für alle Animationsagenten der Gruppe gültig. Es macht beispielsweise wenig Sinn, ein Auto ohne seine Räder zu übertragen. Da Agentknoten und

Hierarchieknoten über eine AABB verfügen, wird im folgenden verallgemeinert vom Animationsagenten gesprochen. Sämtliche Vorgehensweisen in Bezug auf einen Animationsagenten können leicht auf die beiden Knotentypen erweitert werden.

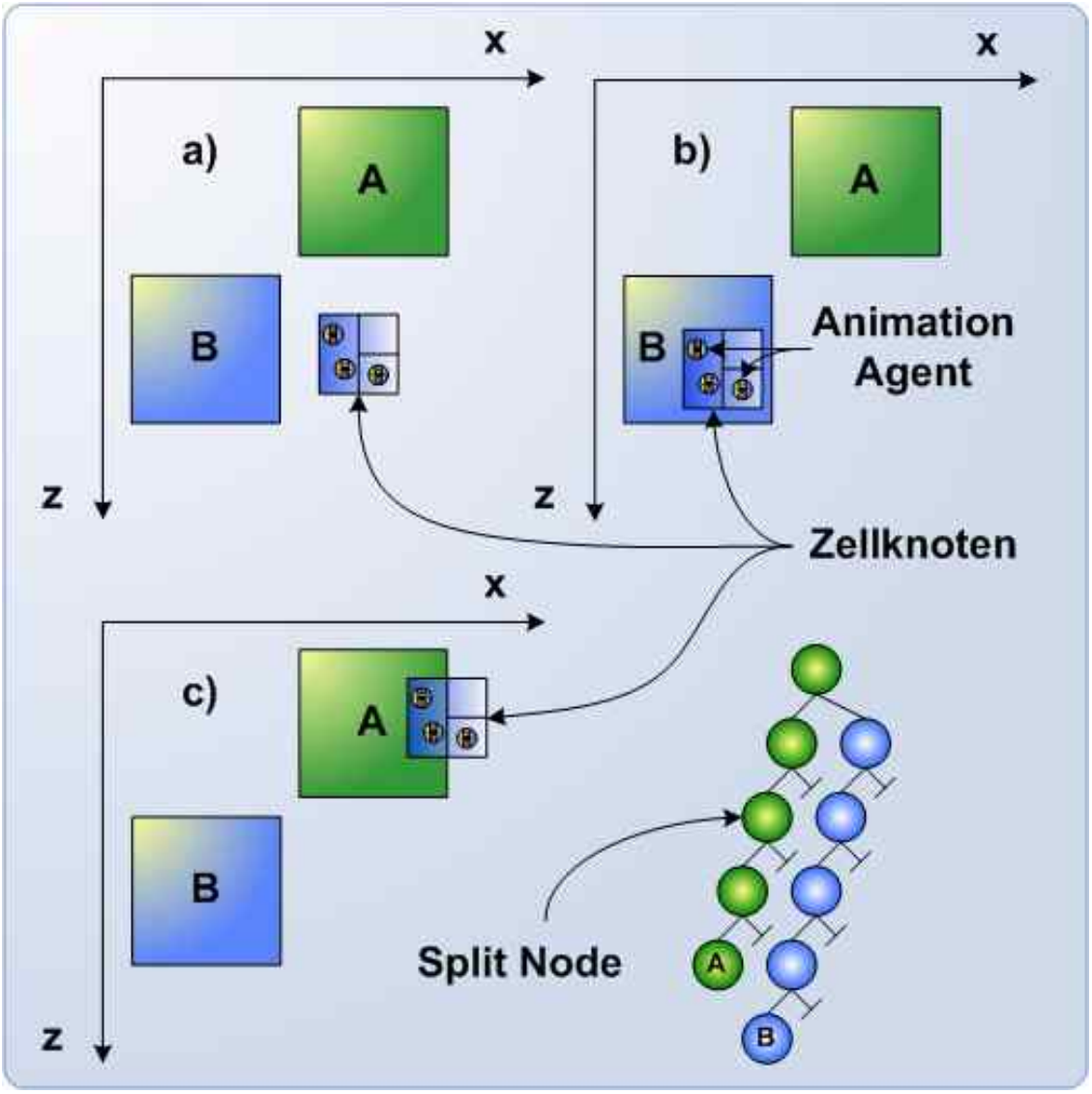

Abbildung 6.27: *Während des Testen der Szenenhierarchie gegen den Clientbaum sind im Falle einer Zelle drei grundlegende Situationen zu unterscheiden. Im ersten Fall liegt die Zelle außerhalb jeglicher Area-of-Interest. Im zweiten Fall erreicht sie einen Clientknoten ohne zuvor unterteilt worden zu sein. Im dritten Fall wird die Zelle unterteilt und es kommt zu einem rekursiven Test der Tochterzellen.*

Beginnend mit der Wurzel der Szenenhierarchie propagiert die Scheduler-Komponente die aktuelle Zelle C durch den Clientbaum. Da jede Zelle einer AABB entspricht, ist der Vorgang ähnlich zum Einfügen der Zonen in Abschnitt 6.5.2: Liegt C vollständig im inneren Halbraum des momentan betretenen Zonenknotens, dann fährt die Traversierung des Clientbaumes mit dem linken Kindknoten fort. Befindet sich C dagegen komplett im äußeren Halbraum, erfolgt eine Traversierung des rechten Teilbaumes. Ansonsten schneidet C die Hyperebene des Zonenknotens und wird in eine innere und äußere AABB unterteilt, welche jeweils zu den entsprechenden Teilbäumen des Zonenknotens propagiert werden. Insgesamt können beim

Propagieren einer Zelle C durch den Clientbaum drei verschiedene Ergebnisse auftreten, die in Abbildung 6.27 dargestellt sind.

Im ersten Fall liegt C außerhalb sämtlicher Clients. Somit besuchen C beziehungsweise alle durch Unterteilungen entstandenen Bereiche C' von C während der Traversierung des Clientbaumes nicht einen einzigen Clientnode. Stattdessen erreichen sie Zonenknoten, in deren äußeren Halbraum sie liegen und die keinen rechten Kindknoten aufweisen. Die räumlichen Kohärenzen der Szenenhierarchie implizieren das gleiche Resultat für alle Unterzellen von C sowie für alle Animationsagenten in C und deren Unterzellen [10].

Im zweiten Fall erreicht C einen Clientnode, ohne zuvor unterteilt worden zu sein. Die Scheduler-Komponente merkt sich den Clientnode und versucht, C durch einen eventuell angehängten Teilbaum des Clientnodes zu propagieren. Beendet C die Traversierung ohne Unterteilung, so bedeutet dies, dass im Falle der Propagierung der Unterzellen von C sämtliche Unterzellen exakt den gleichen Pfad des Clientbaumes durchlaufen würden. Gleiches trifft auch für die Animationsagenten in C sowie in den Unterzellen von C zu. Aus diesem Grund erfolgt kein rekursives Testen der Unterzellen von C, sondern die betreffenden Animationsagenten werden einfach denjenigen Clients zugeordnet, die durch einen der gemerkten Clientnodes auf dem Pfad von C identifiziert sind.

Im dritten Fall erreicht C einen Zonenknoten S, der eine Unterteilung von C verursacht. Die Scheduler-Komponente markiert S und propagiert die resultierenden AABBs C^- und C^+ rekursiv durch die jeweiligen Teilbäume. Betritt C^-, C^+ oder vielleicht gar nur wiederum ein Teilbereich von C^- und C^+ einen Clientnode, so stoppt die Rekursion augenblicklich. Dann nämlich befinden sich alle Unterzellen von C sowie alle Animationsagenten von C und den Unterzellen potentiell innerhalb der Zone eines Clients. Allerdings besteht hierüber keine Gewissheit. Aus diesem Grund müssen zunächst die in C gespeicherten Animationsagenten durch den Clientbaum propagiert werden und anschließend die Unterzellen von C in gleicher Weise, d.h. der Vorgang in Bezug auf C wiederholt sich rekursiv für die Unterzellen. Anstatt dabei aber mit der Wurzel des Clientbaumes zu beginnen, setzt die Scheduler-Komponente den Test am Zonenknoten S fort. Wie schon im zweiten Fall würden nämlich die Unterzellen und Animationsagenten von C den Zonenknoten S ohne eine Unterteilung erreichen. Das Einsetzen der Rekursion mit S hat zur Folge, dass die Unterzellen und Animationsagenten eventuelle Clientknoten vor S nicht erreichen können. Allerdings kann es sich hier nur um Clientknoten handeln, die von C ohne vorhergehende Unterteilung betreten wurden. Derartige Situationen entsprechen dem zweiten Fall, weshalb die Scheduler-Komponente die betreffenden Clientknoten vermerkt hat.

Da die Animationsagenten ebenfalls über eine AABB verfügen, ist ihre grundlegende Behandlung identisch zu den Zellen: Betritt ein Animationsagent oder einer seiner Teilbereiche einen Clientknoten, so ist er allen dort identifizierten Clients zuzuweisen.

Das rekursive Testen der Szenenhierarchie gegen den Clientbaum resultiert nicht nur in der Aussage, ob ein Animationsagent sich in einer Zone befindet. Vielmehr ist das Konzept in der Lage, alle durch einen Animationsagenten berührten Zonen zu erkennen und darüber hinaus

[10] Die Unterzellen repräsentieren die Struktur der Szenenhierarchie. Sie sind also nicht mit Bereichen einer Zelle zu verwechseln, die durch Unterteilungen im Clientbaum entstehen.

die mit den Zonen assoziierten Clients zu ermitteln. Aufgrund der Traversierung der Szenenhierarchie entspricht die Scheduler-Komponente nach Abschnitt 6.3.5 einem Raumrenderer. Die achsenparallele Eigenschaft der Zonen, Zellen und Agenten ermöglicht einfache Tests und Unterteilungen. Die Auswertung der räumlichen Kohärenzen nicht nur in Bezug auf die Szenenhierarchie, sondern auch auf die Zonen selbst, erlaubt eine drastische Reduzierung der Tests zwischen Animationsagenten und Zonen.

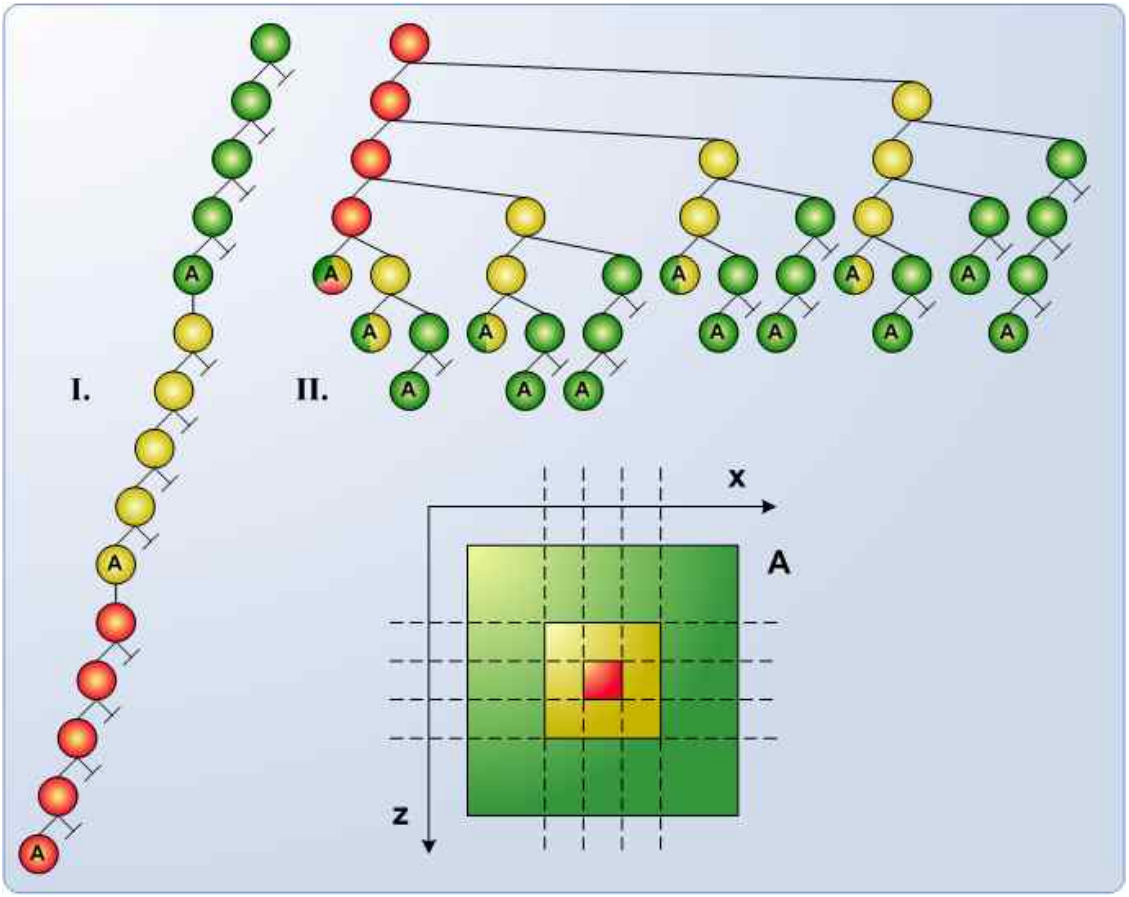

Abbildung 6.28: *Würden die Zonen einer Area-of-Interest von außen nach innen dem Clientbaum hinzugefügt, so ergäben sich die in Bild I.) dargestellten langen Pfade. Im Falle der Einordnung von innen nach außen, käme es zu Situationen wie in Bild II.). Beide Vorgehensweisen haben nach Abschnitt 6.5.4 ihre Nachteile.*

6.5.4 Die Priorisierung der Clientbäume

Da Abschnitt 6.5.2 beim Einfügen eines Clients in den Clientbaum lediglich von einer Zone pro Area-of-Interest ausgeht, fehlt noch eine Aussage, wie den Animationsagenten eine Priorität zugeordnet werden kann. Mit wachsender Zahl der Zonen steigt die Komplexität des Clientbaumes und somit aufgrund der verstärkt auftretenden Unterteilungen der Aufwand des Ansatzes. Das Einfügen aller drei Zonen einer Area-of-Interest impliziert aber nicht allein wegen der Mehrzahl an Zonen Ineffizienzen des Clientbaumes, sondern auch wegen der Verschachtelung der Zonen innerhalb einer Area-of-Interest. Abbildung 6.28 zeigt zwei alter-

native Vorgehensweisen, wie die Zonen einer Area-of-Interest in den Clientbaum eingefügt werden können. Im ersten Fall I.) entspricht die Reihenfolge der Sequenz präventive Zone, dynamische Zone, sichtbare Zone. Hierdurch entsteht ein extremes Beispiel für den in Abschnitt 6.5.2 behandelten Sonderfall sich schneidender Zonen. Als Resultat enthält der Clientbaum sehr lange Pfade mit den sichtbaren Zonen am Ende. Somit müssen sehr viele Entscheidungen getroffen werden, bis ein Animationsagent als innerhalb einer sichtbaren Zone identifiziert wird. Unglücklicherweise ist dies hinsichtlich der Laufzeit die kritischste Information, da die betreffenden Animationsagenten augenblicklich übertragen werden sollten. Wenigstens bietet Fall I.) den Vorteil, dass sobald eine AABB im äußeren Halbraum eines Zonenknotens liegt, der restliche Pfad nicht mehr traversiert werden muss. Ist etwa ein Animationsagent außerhalb der präventiven Zone, so befindet er sich auch außerhalb der beiden übrigen Zonen.

In Alternative II.) werden die Zonen in der umgekehrten Reihenfolge sichtbare Zone, dynamische Zone, präventive Zone in den Clientbaum einsortiert. Hieraus ergeben sich zwar im Vergleich zu I.) kürzere Pfade, dafür müssen aber mindestens vier Zonenknoten betreten werden, bevor eine endgültige Entscheidung über eine AABB getroffen werden kann. So liegt etwa ein Agent außerhalb der sichtbaren Zone nicht zwangsläufig auch außerhalb der präventiven Zone. Aus diesem Grund bietet I.) ein besseres *Best Case* und II.) ein besseres *Worst Case* Verhalten.

Anstatt nun alle drei Zonen einer Area-of-Interest in den Clientbaum einzufügen, verwendet der Ansatz drei verschiedene Clientbäume. Jeder Baum repräsentiert eine bestimmte Priorität, d.h. dem *sichtbaren Clientbaum* werden die sichtbaren Zonen zugeordnet, dem *dynamischen Clientbaum* die dynamischen Zonen und dem *präventiven Clientbaum* die präventiven Zonen. Das in Abschnitt 6.5.3 erläuterte Testen der Szenenhierarchie gegen einen Clientbaum ist in mehrere Schritte unterteilt. Im ersten Schritt wird die Szenenhierarchie in Bezug auf den präventiven Clientbaum analysiert. Sofern sich eine Zelle oder ein Animationsagent innerhalb einer präventiven Zone befindet, erhält der betreffende Knoten der Szenenhierarchie die Priorität der präventiven Zone. Im zweiten Schritt wird die Szenenhierarchie gegen den dynamischen Baum getestet, aber dabei nur diejenigen Pfade traversiert, welche bereits im ersten Schritt mit der präventiven Priorität markiert wurden. Auch hier können räumliche Kohärenzen ausgenutzt werden. Wurde eine Zelle C im ersten Schritt nicht markiert, so ist im zweiten Schritt auch die Traversierung der Unterzellen von C nicht erforderlich. Im dritten Schritt werden die im zweiten Schritt markierten Knoten der Szenenhierarchie gegen den sichtbaren Clientbaum getestet.

Die ersten drei Schritte entsprechen der Initalisierungsphase. Jeder Schritt danach ist durch einen Zyklus geprägt, welcher genau einen Test der präventiven Zonen, u Tests der dynamischen Zonen und v Tests der sichtbaren Zonen beinhaltet, wobei $v > u$. Die Idee der Zyklen basiert auf den beiden folgenden Aspekten:

- Veränderungen innerhalb der sichtbaren Zonen sind kritischer als Vorgänge innerhalb der beiden übrigen Zonen. Gleiches gilt für die dynamischen Zonen in Bezug auf die präventiven Zonen. Als Konsequenz sollten Zonen mit höherer Priorität häufiger überprüft werden als Zonen mit niedrigerer Priorität.

- Die Animationsagenten innerhalb von Zonen mit höherer Priorität stellen im Vergleich zu den Animationsagenten innerhalb von Zonen mit niedriger Priorität nur eine kleine Teilmenge dar. Insbesondere beanspruchen die Animationsagenten innerhalb der präventiven Zonen üblicherweise nur einen geringen Anteil der gesamten Szene. Hieraus folgt, dass der Aufwand für die Überprüfung höher priorisierter Zonen wesentlich geringer ist als für die Analyse niedriger priorisierter Zonen.

6.5.5 Ein Out-of-Core Konzept

Die in Abschnitt 6.5.3 beschriebene Analyse der Szenenhierarchie hinsichtlich des Clientbaumes bietet neben der Minimierung der Vergleiche zwischen Zonen und Animationsagenten noch einen weiteren Vorteil: Während des Tests gegen den Clientbaum besucht die Scheduler-Komponente einen Knoten der Szenenhierarchie maximal einmal. Der Grund hierfür ist, dass sämtliche Zonen in Form des Clientbaumes durch eine einzelne gemeinsame Datenstruktur repräsentiert werden. Bemerkbar macht sich diese Vorgehensweise im Falle einer Out-of-Core Strategie, wie sie in Abschnitt 5.1.1 verlangt wird. Hier besteht nämlich das Risiko, dass jeder Zugriff auf einen Knoten der Szenenhierarchie gleichbedeutend mit dem Einlagern der angeforderten Informationen aus dem Dateisystem in den Hauptspeicher des Servers ist. Ein wiederholtes k-faches hierarchisches Culling der Szenenhierarchie gegen k Zonen wirkt sich dann äußerst negativ auf die Laufzeit aus.

Das grundlegende Konzept der Out-of-Core Strategie ist unabhängig von Server oder Client. Die Hardware beziehungsweise das Betriebssystem stellt einen Speicherraum mit der maximalen Größe κ_{max} zur Verfügung, den das jeweilige Endgerät für die Verwaltung der Szene verwenden darf. Jede Information E_i einer Szene mit $i = 1, \ldots, n$ Einheiten beansprucht einen bestimmten Speicherbedarf L_i, wobei für große Szenen

$$\sum_{i=1}^{n} L_i > \kappa_{max} \tag{6.5}$$

gelten kann. In diesem Fall muss eine Teilmenge der Informationen außerhalb des Hauptspeichers auf einem anderen Medium gehalten werden. Dabei handelt es sich üblicherweise um eine Festplatte, die von der Datei-System-Komponente gekapselt wird. Einige Endgeräte wie etwa PDAs unterstützen derartige lokale Medien nicht und müssen deshalb bei einem Überlauf an Informationen einige der Daten wieder verwerfen. Gleiches gilt auch für Geräte mit einem lokalen Medium, sobald dessen Speichergrenze ebenfalls erreicht ist. Aus diesem Grund kann es sein, dass während einer Session ein Client die gleichen Informationen eventuell mehrfach vom Server anfordert. Auf abstrakte Weise stellt hier also der Server ein externes Medium dar.

Sobald ein Teil der Informationen wegen der Verletzung der Speichergrenze κ_{max} verworfen oder auf ein lokales Medium ausgelagert werden muss, entsteht das Problem, um welche Daten es sich dabei handeln soll. Im umgekehrten Fall, wenn wieder genügend Speicherraum zur Verfügung steht, kommt die Frage auf, welche Informationen denn nun eingelagert werden sollen. Glücklicherweise gibt die Scheduler-Komponente hierauf eine Antwort. Die

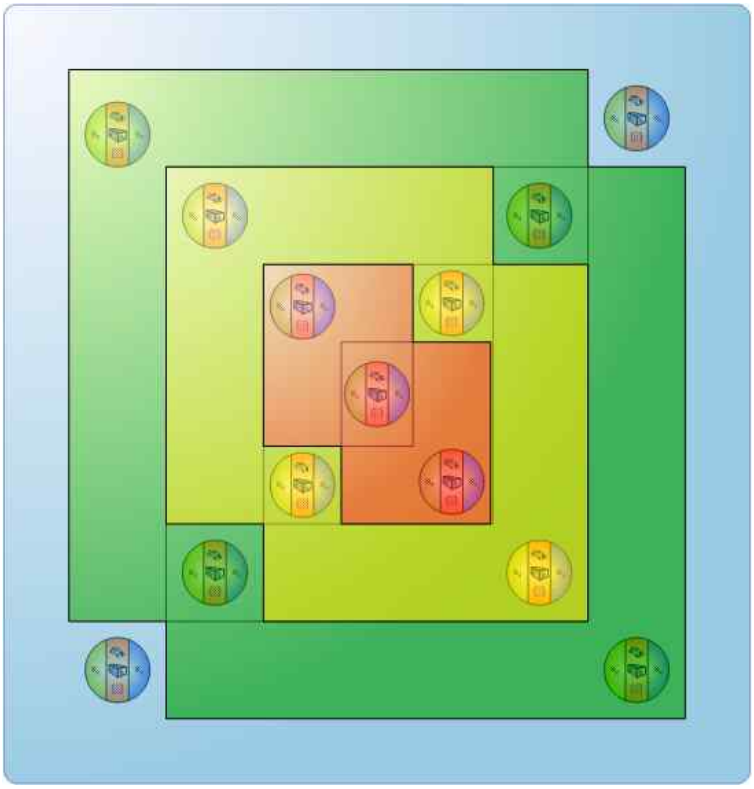

Abbildung 6.29: *Im Rahmen der Out-of-Core Strategie ist es nicht von Bedeutung, welche Clients mit einer Zonen identifiziert sind. Ausschlaggebend ist hier die Zone mit der höchsten Priorität, in der sich ein Animationsagent befindet.*

Priorisierung der Animationsagenten anhand der Areas-of-Interest bietet nämlich weit mehr als lediglich eine Einordnung der Agenten in Bezug auf die einzelnen Betrachterstandorte:

- Animationsagenten, die innerhalb mindestens einer Area-of-Interest liegen, müssen übertragen werden. Höher priorisierte Agenten sind dabei gegenüber niedriger eingestuften Agenten vorzuziehen. Agenten außerhalb jeglicher Area-of-Interest können vernachlässigt werden.
- Nach dem Ansatz von Funkhouser [Fun95] müssen bei der Synchronisation dynamischer Vorgänge nur diejenigen Animationsagenten aktualisiert werden, die innerhalb der Areas-of-Interest liegen. Auch hier gilt die Bevorzugung höher eingestufter Agenten. Ebenso ist die Berücksichtigung von Agenten außerhalb der Zonen überflüssig.
- Die Simulation-Komponenten darf sich, sofern die Simulation es erlaubt, auf die Animationsagenten innerhalb der Areas-of-Interest konzentrieren, da sich nur diese im Sichtbereich eines Clients befinden können.

Alle aufgeführten Aspekte laufen auf einen gemeinsamen Nenner hinaus: Die Informationen, die momentan zu bearbeiten sind, entsprechen den Animationsagenten beziehungsweise deren Daten, die sich aktuell in mindestens einer sichtbaren Zone befinden. An dieser Stelle ist die Identifizierung des mit einer Zone assoziierten Clients nicht mehr von Bedeutung.

Entscheidend ist dagegen, ob der Agent in einer Zone liegt und wenn ja, welche Priorität die Zone repräsentiert. Wie in Abbildung 6.29 dargestellt, zählt im Falle mehrerer Zonen die höchste Priorität. Aufgrund der drei Prioritätsstufen ist nicht nur die Selektion der wichtigsten Informationen, sondern auch deren entsprechende Einordnung möglich. D.h. reicht der Speicherplatz für alle Animationsagenten innerhalb der sichtbaren Zonen aus, so sollte der restliche Raum mit den Agenten der dynamischen Zonen aufgefüllt werden. Daraus ergibt sich folgende Klassifizierung:

- Animationsagenten innerhalb mindestens einer sichtbaren Zone sollten stets im Hauptspeicher gehalten werden. Während einer notwendigen Auslagerung sollten sie erst als letzte berücksichtigt werden, hinsichtlich einer möglichen Einlagerung dagegen als erste.
- Animationsagenten innerhalb mindestens einer dynamischen Zone sollten bei einer notwendigen Auslagerung an vorletzter Stelle berücksichtigt werden, in Bezug auf eine mögliche Einlagerung nach den Animationsagenten der sichtbaren Zonen.
- Animationsagenten innerhalb mindestens einer präventiven Zone sollten bei einer notwendigen Auslagerung an zweiter Stelle berücksichtigt werden, während einer möglicher Einlagerung an dritter Stelle.
- Animationsagenten außerhalb jeglicher Zonen sollten bei einer notwendigen Auslagerung als erste berücksichtigt werden, bei einer möglichen Einlagerung dagegen als letzte.

Obgleich auf Seite der Clients keine Scheduler-Komponente existiert, kann dort das gleiche Konzept angewendet werden. Hier ist allerdings kein Testen gegen einen Clientbaum erforderlich, sondern ein hierarchisches Culling gegen die eigene Area-of-Interest ist vollkommen ausreichend. Da letztere durch die Präsentation-Komponente des Clients bestimmt wird, stehen die notwendigen Informationen bereits lokal zur Verfügung.

6.6 Die Multiplexer- und Demultiplexer-Komponente

Über das in Abschnitt 6.5.4 erläuterte Konzept der Zyklen wertet die Scheduler-Komponente in regelmäßigen Zeitabständen die Prioritäten der einzelnen Animationsagenten aus. Jedesmal wenn sie einen Agenten A innerhalb einer Area-of-Interest identifiziert, übergibt sie die laufende Nummer des aktuellen Zyklus sowie einen Verweis auf A mitsamt der Priorität und dem Client der Area-of-Interest an die Multiplexer-Komponente. Für jeden der auf Seite des Servers registrierten Clients verwaltet die Multiplexer-Komponente eine Art Auftragsliste, welche den Status der aktuell an den Client zu übertragenden Animationsagenten beinhaltet. Entsprechend den Prioritäten der Agenten ist diese Auftragsliste in drei Dringlichkeitsstufen unterteilt. Mit Hilfe der von der Scheduler-Komponenten übergebenen Parameter ist die Multiplexer-Komponenten in der Lage, die Auftragsliste des betreffenden Clients zu selektieren und den Agenten A der korrekten Priorität zuzuordnen. Hierbei sind fünf verschiedene Fälle zu unterscheiden:

- Innerhalb der Auftragsliste des Clients existiert zwar kein Eintrag für A, aber A wurde bereits vorher an den Client übertragen und ist dort momentan verfügbar. Diese Erkenntnis lässt sich aus den serverspezifischen Tabellen der Animationsagenten gewinnen, wie sie beispielsweise in den Abbildungen 6.8 und 6.12 dargestellt sind. Eine derartige Tabelle spezifiziert nicht nur die Clients, auf denen der Agent derzeit existiert, sondern zusätzlich die Identifikation, unter welcher der Agent auf Seite des Clients geführt wird.

- Innerhalb der Auftragsliste des Clients existiert kein Eintrag für A und A wurde noch nicht an den Client übertragen. Aus diesem Grund erstellt die Multiplexer-Komponente einen entsprechenden Eintrag und ordnet ihn der mitgelieferten Priorität zu. Weiterhin vermerkt die Multiplexer-Komponente den Zyklus im Eintrag des Animationsagenten A.

- A existiert bereits in der Auftragsliste des Clients, die Parameter des Schedulers bedeuten aber eine höhere Priorität. Insofern wird A der entsprechenden Priorität zugeordnet. Dies kann auch noch geschehen, wenn zuvor bereits einige Informationen von A an den Client übermittelt wurden. Ein Beispiel hierfür sind progressive Daten. Zusätzlich aktualisiert die Multiplexer-Komponente den Zyklus im Eintrag von A.

- A existiert bereits in der Auftragsliste des Clients und die Parameter der Scheduler-Komponenten entsprechen der dort aufgeführten Priorität. In diesem Fall erneuert die Multiplexer-Komponente lediglich den Zyklus im Eintrag von A.

- A existiert bereits in der Auftragsliste des Clients und die Parameter der Scheduler-Komponenten liegen unter der dort aufgeführten Priorität. Hier erfolgen keinerlei Aktivitäten von Seiten der Multiplexer-Komponente.

Der Grund für das Mitführen der Zyklusnummer eines Eintrages liegt in dessen möglicher Abwertung hinsichtlich seiner Priorität. Erfolgt für A über eine gewisse Zahl von Zyklen entweder gar keine Parameterübergabe durch die Scheduler-Komponente oder enthalten die Parameter nur niedrigere Prioritäten, so wird A um eine Priorität zurückgestuft. Diese Vorgehensweise bietet zwei Vorteile: Zum einen werden Flattereffekte vermieden, bei denen ein Agent beispielsweise durch das permanente Vor- und Zurücklaufen eines Benutzers ständig zwischen zwei Prioritäten hin- und herwechselt. Zum anderen besteht die Möglichkeit, für eine zu übertragende Information auf einfache und kostengünstige Weise eine Priorität zu bestimmen. Mit einem Agenten sind nämlich nicht nur der Agent selbst, sondern, wie in Abschnitt 6.3 dargestellt, eine ganze Reihe anderer Daten verbunden. Hierzu sind beispielsweise der Elementgraph des Agenten sowie die korrespondierenden Pooleinträge zu zählen. Aufgrund der geforderten Speichereffizienz ist es möglich, dass sich mehrere Agenten den gleichen Elementgraphen teilen. Somit kann ein Elementgraph in Bezug auf einen Client mehrere Prioritäten haben. Ausschlaggebend ist letztlich die höchste Priorität. Deren neue Bestimmung ist beim dritten oben aufgeführten Fall einfach möglich und im vierten Fall nicht notwendig. Was aber wenn die Scheduler-Komponente wie im fünften Fall eine niedrigere Priorität für A und damit für dessen Elementgraph meldet? Gilt gleiches dann auch für die anderen Animationsagenten, welche den Elementgraphen referenzieren? Im Zweifelsfall

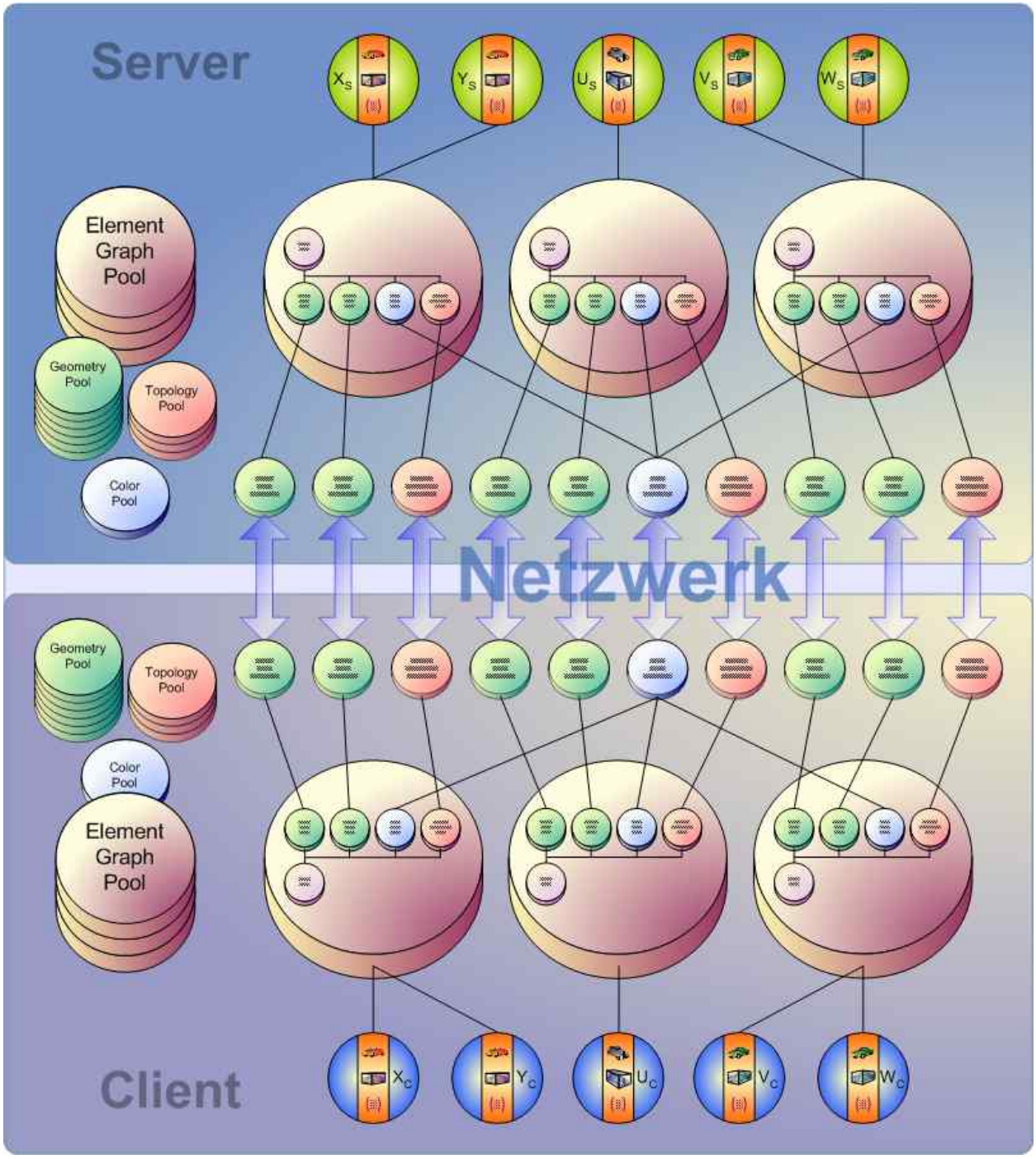

Abbildung 6.30: *Multiplexer und Demultiplexer erstellen jeweils einen Abhängigkeitsgraphen für das Versenden beziehungsweise Empfangen der Daten. Im Falle von Hierarchieknoten sind die Graphen jeweils um eine Stufe zu erweitern. Wichtig ist, dass jeder Knoten des Graphen eine eigene Priorität hinsichtlich der Übertragung besitzt, welche sich aus der maximalen Priorität der mit dem Knoten assoziierten Animationsagenten ergibt.*

müssen die betreffenden Animationsagenten alle überprüft werden, um für den Elementgraphen die aktuelle Prioriät zu bestimmen. Bei entsprechender Zahl der Agenten kann das eine kostspielige Angelegenheit werden.

Abbildung 6.30 verdeutlicht das Problem noch einmal. Mit dem Anmelden eines Animationsagenten in der Multiplexer-Komponente von Seiten der Scheduler-Komponente erstellt erstere für den betreffenden Client einen Abhängigkeitsgraph. Die erste Stufe des Graphen wird durch die an den Client zu übertragenden Animationsagenten repräsentiert. Die zweite Stufe bilden die durch die Agenten referenzierten Elementgraphen. Die dritte Stufe stellen die mit den Elementgraphen assoziierten Pooleinträge dar. Animationsagenten können sich den gleichen Elementgraphen teilen und Elementgraphen wiederum dieselben Pooleinträge. Jeder Knoten des Abhängigkeitsgraphen hat in Bezug auf die Übertragung eine eigene Priorität, die sich mehr oder weniger direkt aus der Priorität der mit dem Knoten verbundenen Animationsagenten ergibt. Die Reihenfolge, in welcher die Knoten an den Client übermittelt werden, ist nicht alleine durch die Prioritäten bedingt, sondern auch durch die beanspruchte Stufe innerhalb des Abhängigkeitsgraphen. So ist beispielsweise ein Elementgraph für einen Client nicht zu gebrauchen, sofern die mit dem Elementgraph verknüpften Pooleinträge nicht ebenfalls vorhanden sind. Aus diesem Grund ergibt sich für die Übertragung einer Szene an einen Client folgende Bearbeitungssequenz:

1. Pooleinträge mit höchster Priorität, d.h. mindestens einer der mit dem Pooleintrag assoziierten Animationsagenten befindet sich in einer sichtbaren Zone.
2. Elementgraphen mit höchster Priorität, d.h. mindestens einer der mit dem Elementgraph assoziierten Animationsagenten befindet sich in einer sichtbaren Zone.
3. Animationsagenten mit höchster Priorität, d.h. der Agent selbst befindet sich in der sichtbaren Zone des Clients.

Auf die oben aufgeführte Sequenz folgen nun in äquivalenter Weise die Informationen mit mittlerer und niedriger Priorität. Diese ist gleichzusetzen mit dem Betreten der dynamischen beziehungsweise präventiven Zone des Clients.

Jeder Knoten des Abhängigkeitsgraphen resultiert in einem Datenstrom, der an den Client gesendet wird. Die Animationsagenten und die Elementgraphen stellen handliche und einfache Datenstrukturen von geringer Größe dar, die problemlos über jede Bandbreite transferiert werden können. Anders sieht die Situation bei den Pooleinträgen aus [11]. Hier ist eine effiziente Kodierung notwendig. Aus diesem Grund beinhaltet jeder Pooleintrag einen Vermerk, der den konkreten Datentyp des Eintrags identifiziert. Über diesen Vermerk ist es möglich, mit Hilfe der in Abschnitt 6.1 eingeführten Codec-Manager-Komponente einen für den Pooleintrag geeigneten Codec zu selektieren. Die Multiplexer-Komponente übergibt dem Codec die zu übertragenden Informationen und sendet diese nach der Kodierung zur Demultiplexer-Komponente des Clients.

Auf der Seite des Clients erstellt die Demultiplexer-Komponente einen ähnlichen Abhängigkeitsgraphen wie die Multiplexer-Komponente. Wegen der oben angeführten Übertragungssequenz erhält sie zunächst die mit einem Animationsagenten assoziierten Pooleinträge. Analog der Multiplexer-Komponente verwendet sie den Datentyp der Pooleinträge, um mit Hilfe der

[11] Mit einem Elementgraph ist zwar ebenfalls ein Pooleintrag assoziiert, allerdings nimmt er in Bezug auf die Übertragung einen gesonderten Status ein.

Codec-Manager-Komponenten einen geeigneten Codec für die Dekodierung der Daten zu finden. Nach den Pooleinträgen trifft in der Regel der Elementgraph ein[12]. Die Elementgraphen versetzen die Demultiplexer-Komponente in die Lage, die eingehenden Datenströme der Pooleinträge zu synchronisieren und einander zuzuordnen. Sobald ein Animationsagent mitsamt seinem Elementgraph und seinen Pooleinträgen vollständig empfangen ist, kann der Agent in die Szene eingefügt und visualisiert werden.

Im Falle der in Abschnitt 6.3.4 erläuterten Hierarchieknoten ist eine Erweiterung der Abhängigkeitsgraphen von Multiplexer- und Demuliplexer-Komponente auf vier Stufen erforderlich. Die Hierarchieknoten nehmen dabei die erste beziehungsweise letzte Stufe des Graphen ein. Somit ist sichergestellt, dass die Agenten einer Animationshierarchie im Kollektiv behandelt werden und auf Seite des Clients nicht lange auf sich warten lassen. Ansonsten müsste das bereits bemühte Auto eventuell für längere Zeit ohne seine Räder auskommen.

Eine besondere Bedeutung kommt Elementgraphen beziehungsweise Pooleinträgen mit progressivem Inhalt zu. Wie in Abschnitt 6.4 erläutert, erhält jeder Knoten eines Elementgraphen während der Simplifizierung einen progressiven Partnerknoten, sofern der Knoten eine Information pro Scheitelpunkt oder pro Dreieck spezifiziert. In diesem Fall referenziert ein Elementgraph in Form des Basismeshes Informationen, die am Stück übertragen werden können als auch progressive Daten, deren Transfer aufgrund ihrer Größe in einzelnen Schritten erfolgen sollte. Die Intention der progressiven Struktur beruht darauf, zunächst das Basismesh zu übermitteln und dieses möglichst schnell auf dem Client anzuzeigen. Erst danach erfolgt eine allmähliche Verfeinerung, wozu der in Abschnitt 6.4 erklärte Ansatz der simultanen Visualisierung zum Einsatz kommen kann. Aufgrund der progressiven Daten ist die oben aufgeführte Übertragungssequenz zu erweitern:

1. Nicht progressive Pooleinträge
2. Elementgraphen
3. Animationsagenten
4. Progressive Pooleinträge

Auch hier gilt weiterhin die Einordnung nach der von der Scheduler-Komponente ermittelten Dringlichkeitsstufe.

Im Falle eines Animationsagenten mit progressiven Daten werden also an erster Stelle die nicht progressiven Pooleinträge übermittelt. Hierzu zählen alle Informationen des Basismeshes einschließlich eventueller Zusatzattribute. Danach folgt der Elementgraph sowie der Animationsagent selbst. Die Demultiplexer-Komponente empfängt die eingehenden Pooleinträge, welche für sie noch keine besondere Bedeutung haben. Dies ändert sich erst mit dem Erhalt des Elementgraphen. Anhand der dort eingetragenen progressiven Partnerknoten erkennt die Demultiplexer-Komponente, dass es sich bei den bereits empfangenen Pooleinträgen um Informationen eines Basismeshes handelt und deshalb weitere Datenpakete folgen

[12]Bedingt durch Übertragung und paralleler Bearbeitung können in der Praxis auch zufällige Reihenfolgen entstehen.

werden. Somit stellt die Demultiplexer-Komponente über die progressiven Partnerknoten die Verbindung zwischen jeweils zwei getrennten Datenströmen her, um später den progressiven Datenstrom an den bereits abgeschlossenen Datenstrom des Basismeshes anfügen zu können. Wenn alle nicht progressiven Informationen des Elementgraphen eingegangen sind, dann entfernt die Demultiplexer-Komponente aus dem Elementgraphen die progressiven Partnerknoten. Weil die Verknüpfung zum Basismesh bereits erfolgt ist, stellen diese Knoten nur noch Ballast dar, der die Effizienz der Elementgraphrenderer mindern würde. Anschließend erklärt die Demultiplexer-Komponente den Elementgraphen für visualisierbar und trägt mit Erhalt des Animationsagenten selbigen in die Szene ein.

Beim späteren Empfang progressiver Datenpakete ist die Synchronisation der einzelnen Datenströme von großer Bedeutung. Beispielsweise kann es passieren, dass mehr Normalen als Scheitelpunkte eintreffen. Mittels des in Abschnitt 6.4 erläuterten progressiven Formats stellt dies noch kein Problem dar, weil alle Datenpakete einfach an ihre Vorgänger im jeweiligen Datenstrom angefügt werden können. Eine Ausnahme bilden topologische Informationen, die einer gesonderten Behandlung bedürfen, um die durch die Kantenkollabierung entfernten Dreiecke wieder zu rekonstruieren. Hier besteht nun die Gefahr, dass in die topologische Information mehr Dreiecke eingefügt werden, als an Scheitelpunkten oder Zusatzattributen vorhanden ist. D.h. die Topologie beinhaltet Indizes, die ins Leere weisen. Die Lösung des Problems basiert auf den in Abschnitt 6.4 eingeführten Dependencies. Eine Dependency beschreibt exakt, welche weiteren Datenströme mit einem Topologieknoten des Elementgraphen verbunden sind. Die Demultiplexer-Komponente kann die Dependencies eines Elementgraphen einfach über das Rendern des Graphen identifizieren. Bevor die Demultiplexer-Komponente Dreiecke in die Topologie einfügt, bestimmt sie innerhalb der Dependency den Pooleintrag mit der niedrigsten Zahl an Informationen. Anschließend darf sie solange Dreiecke hinzufügen, bis der Index eines Dreiecks den Grenzwert überschreitet.

Neben der Organisation der Datenübertragung obliegt es der Multiplexer-Komponente, die Menge der progressiven Daten zu definieren, die ein Client erhalten soll. Die Multiplexer-Komponente berücksichtigt dazu die in der Client-Datenbank-Komponente eingetragenen Leistungsmerkmale des Clients. Diese werden zusammen durch einen einzelnen Benchmarkindex mit einem normierten Wertebereich zwischen Null und Eins repräsentiert. Der Benchmarkindex wird mit einem ähnlichen Index für die aktuelle Bandbreite der Verbindung zwischen Server und Client kombiniert und anschließend als prozentualer Faktor für die progressive Datenmenge verwendet. Ein resultierender Index von 0.5 bedeutet also den Transfer von 50% der progressiven Daten auf Seite des Clients.

Die Demultiplexer-Komponente erfährt von der Multiplexer-Komponente, wieviele progressive Informationen sie zu erwarten hat. Sie verwendet den Wert, um verschiedene Schwellwerte für die Erzeugung fester Detailstufen zu bestimmen. Im Konzept der vorliegenden Arbeit handelt es sich dabei um fünf Detailstufen, die über Prozentanteile definiert sind. Jedesmal wenn die Anzahl der eingefügten Dreiecke einen der Schwellwerte überschreitet, definiert die Demultiplexer-Komponente eine entsprechende Detailstufe. Da für eine Detailstufe letztlich nur die Anzahl der Dreiecke von Bedeutung ist, beinhaltet lediglich der Pooleintrag mit den topologischen Informationen eine Information über die Detailstufen.

6.7 Die Präsentation-Komponente

Wie in Abschnitt 6.1 erläutert fallen der Präsentation-Komponenten mehrere Aufgaben zu. Ein Schwerpunkt liegt dabei in der Interaktion mit dem Benutzer, wozu nicht nur mögliche Eingaben des Benutzers zu zählen sind, sondern auch die Visualisierung der auf dem Client momentan vorhandenen Szene. Selbige wird in Abschnitt 6.7.2 behandelt. Da gemäß den Anforderungen in Abschnitt 5.1.2 eine allgemeine Navigationsschnittstelle verlangt ist, beschreibt Abschnitt 6.7.3 eine entsprechende Lösung. Bedingt durch die zentrale Rolle bei der Kommunikation mit dem Benutzer, besteht eine weitere Aufgabe der Präsentation-Komponente in der Prekalkulation des Navigationspfades. Dieser Teil ist im folgenden Abschnitt 6.7.1 erläutert.

6.7.1 Prekalkulation und Adaption der Area-of-Interest

Die Präsentation-Komponente ist Teil des in Abschnitt 6.5.1 beschriebenen Area-of-Interest Konzepts. Sie ermittelt anhand der Bewegungen des Benutzers die Dimensionen der einzelnen Zonen und überträgt diese nach Abschnitt 6.2 an die Client-Datenbank-Komponente des Servers. Dort kann die Scheduler-Komponente die Area-of-Interest des betreffenden Clients auslesen und die Animationsagenten innerhalb der jeweiligen Zonen identifizieren. Die Messungen der Präsentation-Komponente in Bezug auf die Navigationsparameter erfolgen unabhängig von einer eventuellen Simulation auf Seiten des Servers in regelmäßigen Zeitabständen. Ein Transfer der berechneten Area-of-Interest zur Client-Datenbank-Komponente tritt nur im Falle eventueller Änderungen auf. Das Zeitintervall zwischen zwei Messungen orientiert sich nicht an den Interaktionen des Benutzers, da selbiger ansonsten ein zu hohes Informationsaufkommen provozieren könnte [13]. Als Konsequenz können innerhalb eines Zeitintervalls durchaus mehrere Benutzereingaben zum Tragen kommen.

Teler et al. [TL01] beschreiben in ihrer Arbeit einen Ansatz zur Prekalkulation der Bewegungen eines Betrachters. Sie definieren eine Navigation als eine Kombination aus wenigen grundlegenden Situationen, nämlich dem Verharren auf der Stelle, dem Drehen im Kreis und dem geradlinigen Vorwärts- beziehungsweise Rückwärtsschreiten. Eventuelle Kurvenverläufe approximieren sie über mehrere geradlinige Bewegungen. Ihr Konzept beruht auf der Aussage, dass ein Betrachter mit hoher Wahrscheinlichkeit eine einmal begonnene Form der Navigation kontinuierlich fortsetzt. Befindet sich der Betrachter beispielsweise zum Zeitpunkt t_i in einer geradlinigen Vorwärtsbewegung, so trifft gleiches mit ziemlicher Sicherheit auch für den Zeitpunkt t_{i+1} zu. Für die Überprüfung des Konzepts von Teler et al. wurde im Rahmen der vorliegenden Arbeit ein Feldtest mit 30 Probanden durchgeführt. Ziel war die Analyse des Verhaltens der Benutzer in verschiedenen 3D Applikationen. Innerhalb des Feldtests handelte es sich dabei um einen Ego Shooter, einen Tactical Shooter, ein 3D Modellierungswerkzeug sowie einem Programm für die Betrachtung großflächiger 3D Geodaten beziehungsweise 3D Landschaftsmodelle. Abbildung 6.31 illustriert einen Teil der Ergebnisse des Feldtests. Demnach liegt der Anteil bogenförmiger Bewegungen bei 50%, weshalb

[13] Zum Beispiel könnte ein aggressiver Benutzer versuchen, einen Server absichtlich durch das zehntausendfache Übertragen seiner Area-of-Interest pro Sekunde lahmzulegen.

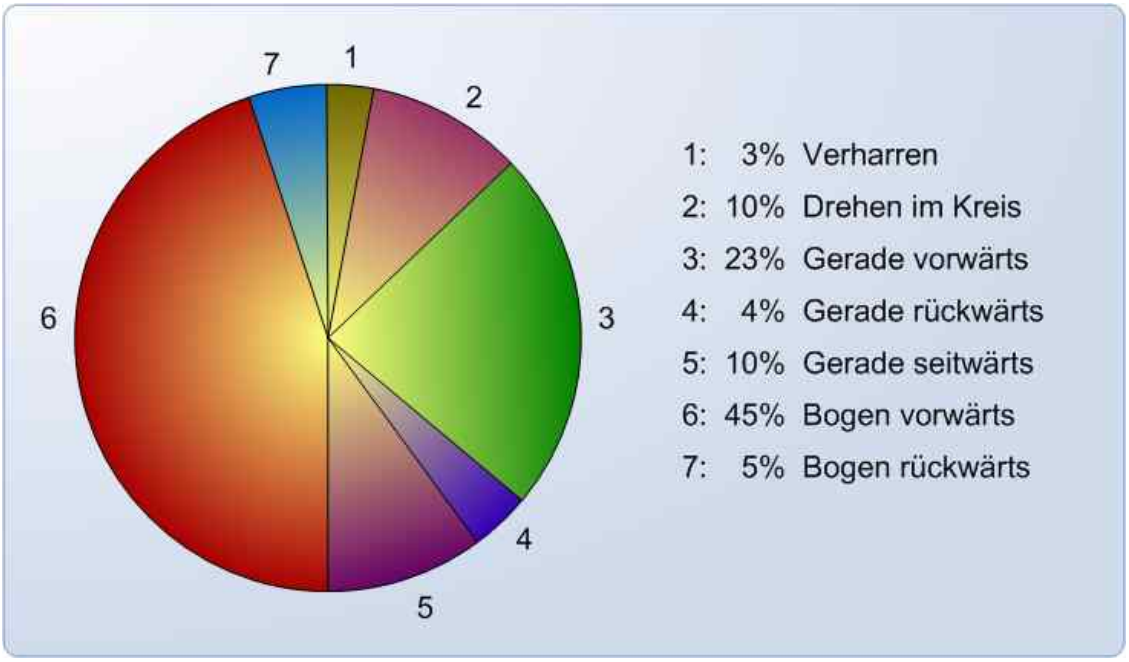

Abbildung 6.31: *Ein Teil der Ergebnisse aus einem Feldtest, in welchem das Verhalten der Benutzer während der Bedienung verschiedener 3D Applikationen untersucht wurde. Entscheidender Punkt ist hier, dass sich eine Navigation gemäß Teler et al. [TL01] in wenige grundlegende Situationen diskretisieren lässt.*

eine mitunter doch sehr ungenaue Approximation über geradlinige Vorgänge nicht ratsam erscheint. An dieser Stelle erfolgt daher eine andere Klassifizierung der Bewegungen, gleichwohl die grundlegende Annahme äquivalent zu Teler et al. ist. Auch hier wird davon ausgegangen, dass ein Benutzer mit hoher Wahrscheinlichkeit eine Bewegungsform kontinuierlich beibehält. Die Klassifzierung ergibt sich wie folgt:

- **Positionsbezogene Situationen**: Hierunter sind das Verharren auf der Stelle sowie das Drehen im Kreis zu verstehen. Insgesamt beanspruchen sie 13% der Bewegungen.
- **Geradlinige Situationen**: Derartige Situationen umfassen geradlinige Vorwärts-, Rückwärts- und Seitwärtsbewegungen. Sie besitzen einen Anteil von 37%.
- **Bogenförmige Situationen**: Hierzu sind bogenförmige Vorwärts- und Rückwärtsbewegungen zu zählen. Ihr Anteil liegt bei 50%. Entsprechende Seitwärtsbewegungen werden von den meisten Applikationen nicht unterstützt.

Im verbleibenden Teil des Abschnitt wird nun die Anpassung der Area-of-Interest an die oben aufgeführten Situationen beschrieben. Wie schon in Abschnitt 6.5 illustrieren sämtliche Abbildungen lediglich den zweidimensionalen Fall. Die hier als Indizes verwendeten Bezeichnungen l, r, o, u stehen immer für *links*, *rechts*, *oben* sowie *unten* und referenzieren den entsprechenden Rand einer Zone.

Abbildung 6.32 zeigt die Behandlung positionsbezogener Situationen. Bild I.) entspricht dem Betreten einer Szene durch den Benutzer. Der Betrachter steht im Zentrum der sichtbaren

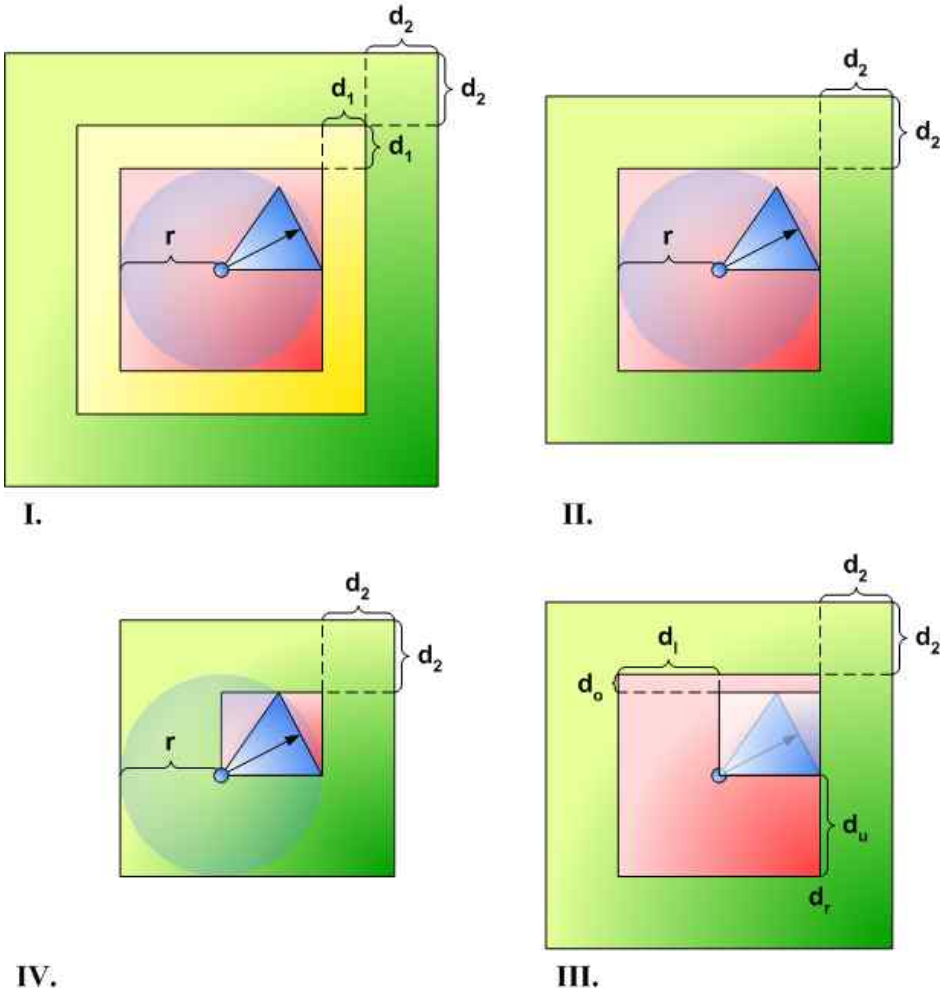

Abbildung 6.32: *Die Anpassung der Area-of-Interest im Falle positionsbezogener Aktionen. Eine dynamische Zone ist nur erforderlich, sofern sich der Benutzer bewegt. Folglich verkleinert sich die dynamische Zone schrittweise mit andauernder Beibehaltung der Position.*

Zone, wobei sich der Abstand des Zonenrandes zum Betrachterstandort aus dem Radius r ergibt. Nach Abbildung 6.23 resultiert r aus der Distanz zwischen Augpunkt und einem beliebigem Scheitelpunkt der hinteren Clippingebene. Sofern der Benutzer seine augenblickliche Position beibehält, kann die Sichtpyramide niemals die sichtbare Zone verlassen. Der Grund für die zentrale Lage des Betrachters mit dem Abstand r rührt aus der Beobachtung her, dass viele Benutzer zu Beginn sich zunächst einen Überblick der Szene verschaffen wollen und sich deshalb einmal im Kreis drehen. Der Abstand der dynamischen sowie der präventiven Zonen ist jeweils relativ zueinander durch die Defaultdistanzen d_1 und d_2 definiert. Hiermit wird ein plötzlicher Positionswechsel des Betrachters nach der Orientierungsphase berücksichtigt. Solange der Benutzer allerdings auf der aktuellen Position verweilt, ist eine dynamische Zone überflüssig. Aus diesem Grund wird der Abstand zwischen sichtbarer und dynamischer Zone mit fortlaufender Zeitdauer reduziert. Im Extremfall besteht die Area-of-Interest wie in Bild II.) zu sehen nur noch aus zwei Zonen. Das weitere Vorgehen hängt davon ab, ob der Benutzer bewegungslos verharrt oder sich auf der Stelle im Kreis dreht. Wohingegen letzteres keine Veränderung der aktuellen Area-of-Interest verursacht, impliziert ersteres eine Verklei-

nerung der sichtbaren Zone. Nach Bild III.) wird zunächst die AABB der Sichtpyramide und darauf für jede Kante der AABB die Distanz zur korrespondierenden Kante der sichtbaren Zone ermittelt [14]. Diese Distanzen sind als d_l, d_r, d_o sowie d_u verzeichnet, wobei d_r im dargestellten Fall dem Wert 0 entspricht. Ähnlich der dynamischen Zone werden die Abstände d_l, d_r, d_o, d_u mit zunehmender Dauer des Verharren auf 0 reduziert. In Bild IV.) entspricht die sichtbare Zone schließlich der AABB der Sichtpyramide. Da sich die Dimensionen einer Zone relativ aus den Distanzen zu der benachbarten, umschlossenen Zone ergeben, sinken die Ausmaße der präventiven Zone ebenfalls. Wie in Bild IV.) angedeutet, gibt es hierzu allerdings zwei Bedingungen: Zum einen darf der Rand der präventiven Zone nicht näher als der Radius r zum Betrachterstandort kommen. Zum anderen darf der Abstand zwischen sichtbarer und präventiver Zone nicht unter d_2 fallen. Die Berücksichtigung des ersten Kriteriums trägt einem plötzlichen Drehen des Betrachters um die eigene Achse Rechnung. Die zweite Bedingung erlaubt eine Vorwärtsbewegung des Betrachters im ungefähren Bereich der Blickrichtung. Dieser Fall ist der wahrscheinlichste nach der Orientierungsphase.

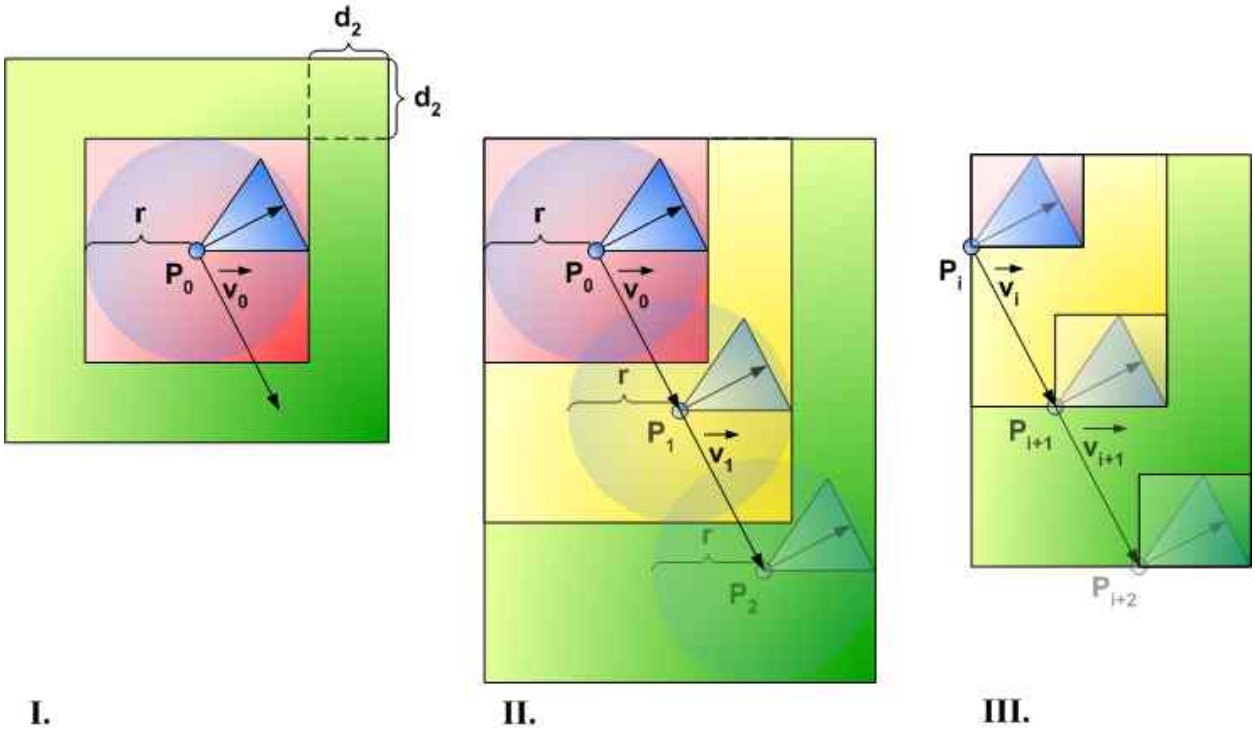

Abbildung 6.33: *Geradlinie Bewegungen gehören zu den einfacheren Positionswechseln, da die Transformation über einen Vektor dargestellt werden kann. Ähnlich den positionsbezogenen Handlungen nähern sich die Zonen mit fortlaufender Dauer der Bewegung an die AABB der Sichtpyramide.*

Geradliniges Navigieren zählt zu den einfachsten Situationen möglicher Positionswechsel, da sich die Bewegungen mit Hilfe eines zeitabhängigen Richtungsvektors beschreiben lassen. Jede geradlinige Bewegung beginnt mit dem Zeitpunkt t_0 und endet mit dem Zeitpunkt t_n, sobald der Betrachter in eine andere Form der Navigation überwechselt. Analog den Animationen und der dortigen Animationszeiten (siehe Abschnitt 6.3.3) hat jede Bewegung

[14] Im dreidimensionalen Fall sind die Kanten auf die Flächen der AABBs zu erweitern.

somit eine eigene *Navigationszeit*, welche über die zeitlichen Messabstände der Präsentation-Komponente diskretisiert ist. Ein Richtungsvektor $\vec{v_i}$ mit $i = 1, \ldots, n$ korrespondiert zu dem Zeitpunkt t_i der Navigationszeit und repräsentiert den Wechsel von der Position P_i zu der Position P_{i+1}. Die Länge von $\vec{v_i}$ ist gleichbedeutend mit der Geschwindigkeit des Betrachters und beschreibt in Kombination mit anderen Richtungsvektoren eine eventuelle Beschleunigung des Betrachters. Grundsätzlich muss zwischen zukünftigen und bereits vergangenen Bewegungen unterschieden werden. Entspricht die aktuelle Navigationszeit dem Zeitpunkt t_a, dann versucht die Präsentation-Komponente anhand der Positionen P_k und der Richtungsvektoren $\vec{v_k}$ mit $0 \leq k \leq a$ eine Abschätzung der zu erwartenden Positionen P_{a+1} und P_{a+2}. Erreicht die Navigationszeit den Zeitpunkt t_{a+1} beziehungsweise t_{a+2}, so wird die jeweilige vorhergesagte Position durch den nun wirklich eingenommenen Standort des Betrachters ersetzt. Diese Werte bilden dann die Basis für weitere Annahmen. Die Anzahl der vorhergesagten Positionen ist dadurch bedingt, dass P_{a+1} die dynamische Zone und P_{a+2} die präventive Zone spezifiziert. Abbildung 6.33 illustriert hierzu eine geradlinige Seitwärtsbewegung nach der Orientierungsphase des Benutzers in Bild I.). Die Präsentation-Komponente ermittelt anhand der Benutzereingaben zum aktuellen Zeitpunkt t_0 einen Richtungsvektor $\vec{v_0}$ und damit ausgehend von P_0 eine zu erwartende Position P_1. Ob der Betrachter P_1 auch wirklich erreicht, ist nicht sicher, da zwischen t_0 und t_1 noch weitere Interaktionen des Benutzers erfolgen können. Zudem sind vor dem Zeitpunkt t_0 keine stabilisierenden Angaben erhältlich. Aus diesem Grund wird in Bild II.) um P_1 ein Puffer mit dem Radius r gelegt. Damit ist einer geringfügigen Änderung sowohl der Bewegungs- als auch der Blickrichtung stattgegeben. Die dynamische Zone ergibt sich aus der sichtbaren Zone sowie der durch P_1 und r aufgespannten AABB. Äquivalentes gilt für die präventive Zone in Hinsicht auf P_2. Da eine Beschleunigung zum Zeitpunkt t_0 noch nicht abgeschätzt werden kann, gilt $\vec{v_0} = \vec{v_1}$. Sobald der Benutzer die geradlinige Bewegung über einen bestimmten Zeitraum einhält, erfolgt nach Bild III.) eine ähnliche Anpassung der sichtbaren Zone wie im Falle der positionsbezogenen Situationen: Die sichtbare Zone reduziert sich mit zunehmender Zeitdauer auf die AABB der Sichtpyramide. Dasselbe trifft ebenfalls auf die Pufferbereiche um die erwarteten Positionen zu. Aufeinander folgende Richtungsvektoren können nun unterschiedliche Beträge aufweisen.

Eine wichtige zu klärende Frage ist, welche Konsequenzen die Verwendung der einzelnen Zonen hinsichtlich des Positionswechsels eines Betrachters mit sich bringen. Zu einem Zeitpunkt t_i sollten bereits alle Animationsagenten innerhalb der sichtbaren Zone auf dem Client vorhanden sein. Durch die Bewegung nach P_{i+1} entsteht eine dynamische Zone, deren Inhalt nun die höchste zu übertragende Priorität darstellt. Während des Zeitintervalls zwischen t_i und t_{i+1} visualisiert der Client möglicherweise mehrere Frames, d.h. der nächste Frame wird nicht zwangsläufig erst zum Zeitpunkt t_{i+1} bearbeitet. Das wiederum bedeutet, dass der Betrachter von P_i nach P_{i+1} theoretisch alle auf diesem Weg liegenden Animationsagenten sehen kann. Es ist also nicht ausreichend, lediglich die von den Positionen P_i und P_{i+1} aus sichtbaren Agenten zu übertragen. Aus diesem Grund sollte im optimalen Fall die Übertragung der Animationsagenten innerhalb der dynamischen Zone bereits abgeschlossen sein, bevor der Betrachter selbige auch nur mit der Sichtpyramide streift. Ansonsten können während der Visualisierung entweder Lücken bemerkbar sein oder aus dem Nichts neue Animationsagenten erscheinen. Dieser Umstand unterstreicht die Bedeutung der präventiven Zone, denn mit einer gewissen Wahrscheinlichkeit ist der erforderliche Transfer zumindest

teilweise schon aufgrund der betreffenden Zone geschehen. Eine Garantie kann in der Praxis aber nicht gegeben werden. Ein Kriterium wie im Falle von Teler et al. mit Gleichung 5.4 ist ebenso wenig umsetzbar wie eine kontinuierliche Priorisierung der Animationsagenten beispielsweise anhand der Entfernung zum Betrachter. Ersteres scheitert unter anderem an der schwankenden Bandbreite. Eine feine Priorisierung, wie sowohl bei Teler et al. als auch bei einem stufenlosen Distanzkriterium gegeben, würde implizieren, dass der Server in Hinsicht auf eventuell mehrere hundert Clients ständig die Prioritäten der Animationsagenten überprüfen und vor allem sortieren müsste. Allein dieser Aufwand könnte schon eine Visualisierung in Echtzeit auf den Clients verhindern. Das Konzept der drei Zonen bietet hier zwar eine gröbere, dafür aber realistischere Auflösung.

Sobald die Navigationszeit t_{i+1} erreicht, wird die sichtbare Zone von P_i nach P_{i+1} verschoben. Frames zwischen t_i und t_{i+1} erhalten hierdurch keine Berücksichtigung von Seiten der sichtbaren Zone und der mit ihr verbundenen Priorität. Einerseits ist dies nicht möglich, weil innerhalb des Zeitintervalls keine Messung der Betrachterparameter erfolgt und der Server deshalb keine Kenntnis von der genauen Lage der sichtbaren Zone hat. Andererseits ist es aber auch gar nicht notwendig, da die dynamische Zone bereits die höchste Priorität repräsentiert. Das Verschieben der sichtbaren Zone macht aber Sinn, sofern der Benutzer an P_{i+1} stehen bleibt oder auch nur kurz zögert. In diesem Fall sollten sich nämlich eventuelle Lücken im Sichtbereich augenblicklich schließen, d.h. die Animationsagenten sollten dort eine höhere Priorität erhalten als mögliche verbleibenden Reste der dynamischen Zone.

Die soeben erläuterten Prinzipien gelten sämtlich auch für die Behandlung bogen- beziehungsweise kurvenförmiger Bewegungen. Eine derartige Navigation lässt sich frühestens nach drei Messungen der Betrachterparameter detektieren. Eine Eigenschaft nicht geradliniger Bewegungen liegt darin, dass die Betrachter fast immer die Blickrichtung der Bewegungsrichung anpassen. Abbildung 6.34 zeigt eine Kurve durch die vergangenen Betrachterpositionen P_{i-3}, P_{i-2}, P_{i-1} bis hin zum Standort P_i des aktuellen Zeitpunkts t_i. Theoretisch ist eine Vorhersage von P_{i+1} durch die in Abschnitt 4.1.4 erwähnten Interpolationsschemata möglich. Allerdings ergeben sich hierdurch nicht zuletzt aufgrund der Oszillation recht gewagte Prognosen. Abbildung 6.34 macht dies durch das gestrichelte Segment am Ende der Kurve deutlich. Stattdessen geht der vorliegende Ansatz von einer Fortführung der Navigation zwischen den Positionen P_{i-2}, P_{i-1} sowie P_i aus. Werden die Positionen als Ortsvektoren interpretiert, so bilden die Vektoren $\vec{a}$ und $\vec{b}$ mit $a = P_{i-1} - P_{i-2}$ beziehungsweise $b = P_i - P_{i-1}$ die Basis für eine Drehebene h. Die Drehachse entspricht der Normalen von h, deren Schnittpunkt mit h zu P_{i-2}, P_{i-1} sowie P_i den gleichen Abstand hat. Auf diese Weise entsteht der in Abbildung 6.34 illustrierte Kreis, der durch die Positionen P_{i-2}, P_{i-1}, P_i verläuft. Der Ansatz impliziert, dass die folgenden Positionen P_{i+1} und P_{i+2} ebenfalls auf dem Kreis liegen. Sie können unter Berücksichtigung der Winkelgeschwindigkeit und eventueller Beschleunigungen ermittelt werden.

Das hier beschriebene Konzept zur Adaption der Area-of-Interest an die Navigation des Betrachters stellt einen Kompromiss zwischen dem Aufwand für die Selektion der zu transferierenden Animationsagenten und deren eigentlicher Übertragung dar: Auf der einen Seite ist die Identifizierung der betreffenden Animationsagenten sehr schnell möglich, auf der anderen Seite werden aber eventuell Agenten übermittelt, welche der Benutzer niemals zu Gesicht bekommt. Die Anzahl dieser Animationsagenten wird im folgenden als *Verschwendungsrate*

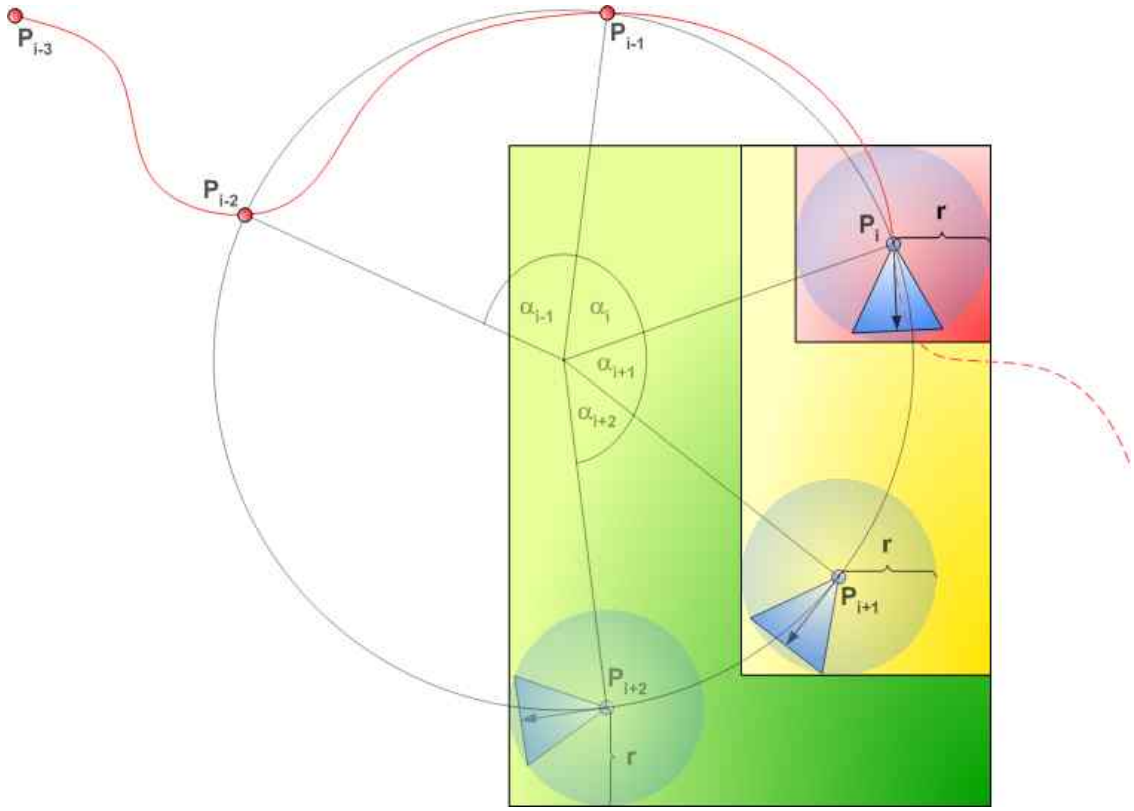

Abbildung 6.34: *Im Falle kurven- beziehungsweise bogenförmiger Bewegungen wird anhand der letzten drei festgestellten Betrachterpositionen eine Drehebene definiert. Der Ansatz geht von einer kontinuierlichen Fortsetzung der Bewegung aus und bestimmt daher zukünftige Standorte auf dem Kreis, der durch die drei letzten Positionen verläuft.*

bezeichnet, deren Minimierung das Ziel der Adaption ist. Eine Präzision wie im Falle eines Occlusion Culling Verfahrens kann in Szenen mit hoher Verdeckungstiefe nicht erreicht werden. Derartige Techniken sind jedoch viel zu rechenintensiv und verlieren zudem mit sinkender Verdeckungstiefe drastisch an Effizienz. Um alle Anforderungen aus Abschnitt 5.1.1 bezüglich der Area-of-Interest zu erfüllen, ist noch die Behandlung dynamischer Animationsagenten notwendig. Eine Voraussetzung hierzu bildet die konsistente Repräsentation des Raumunterteilungsbaumes, wofür Abschnitt 6.3.4 eine Lösung vorschlägt.

Grundsätzlich gibt es für die Identifizierung dynamischer Agenten zwei Möglichkeiten. In der ersten Option werden die Dimensionen der Zonen an die Geschwindigkeit der Animationsagenten angepasst. Hohe Geschwindigkeiten implizieren dabei größere Zonen. Leider ist die Berechnung der Zonen auf diese Weise äußerst schwierig, da meistens nicht klar ist, welche Geschwindigkeiten maximal in einer Szene auftreten können und welche davon eine Zone wirklich betreffen. Die bessere Alternative ist daher, die Animationsagenten anhand der in Abschnitt 6.3.4 erläuterten Temporary Bounding Box in den Raumunterteilungsbaum einzusortieren. Da eine derartige Vorgehensweise auch die dort diskutierten Nachteile hat, sollten die gültigen Zeitintervalle der Bounding Boxes möglichst klein gewählt werden.

6.7.2 Ein effizientes Occlusion Culling Verfahren

Ein Teil der Kommunikation zwischen Präsentation-Komponente und Benutzer beruht neben möglichen Eingaben auch in der Visualisierung der auf dem Client vorhandenen Szene. Gemäß der Anforderungen in Abschnitt 5.1.4 bietet das Konzept der Renderer eine flexible Möglichkeit, verschiedene Formen der Ausgabe umzusetzen. Darüber hinaus kapseln die Renderer aber auch verschiedene Techniken, die während der Visualisierung zum Einsatz kommen. Beispielsweise ist es möglich, einen Renderer nach einer View Frustum Culling Strategie zu konzipieren und ihn dann zur Laufzeit durch einen Occlusion Culling Renderer zu ersetzen. Da die Renderer Teil der Plattform sind, kann konsequenterweise für Plattformen mit entsprechender Grafik-Hardware der Algorithmus von Bartz et al. [BMH99] in einem Renderer verwirklicht werden. Auf Plattformen ohne HP-Flag und NVIDIA-Occlusion-Query ist dann die Verwendung eines anderen Renderers erforderlich. In der Regel entsprechen Renderer zur Ausgabe der Szene einem Raumrenderer, welcher den Raumunterteilungsbaum traverisert. In Abhängigkeit von als sichtbar identifizierten Animationsagenten wird dann ein Elementgraphrenderer aufgerufen, welcher den mit dem Agenten assoziierten Elementgraphen bearbeitet.

In diesem Abschnitt wird nun ein hardwarebasiertes Occlusion Culling Verfahren vorgestellt, welches die Anforderungen aus Abschnitt 5.1.4 erfüllt und somit keine Erweiterungen benutzt, die über den Standard aktueller Grafikbibliotheken hinausgehen. Ähnlich Abschnitt 6.5.3 wird anstelle von Agentknoten und Hierarchieknoten verallgemeinert von Animationsagenten gesprochen, da beide Knotentypen des Raumunterteilungsbaumes über eine AABB verfügen. Die AABB des Hierarchieknoten kann allerdings dazu verwendet werden, um alle mit der Hierarchie verbundenen Animationsagenten als unsichtbar zu identifizieren. Insofern macht sich auch hier das Konzept der Hierarchieknoten bezahlt.

Das folgende Konzept [SSB03] basiert auf der Auswahl von Occludern und ist unterteilt in zwei Phasen. Erstere entspricht einem View Frustum Culling und letztere einem Occlusion Culling. Das Occlusion Culling selbst ist nochmals gegliedert in die Selektion der Occluder, in deren Aufbereitung sowie in die abschließende Sichtbarkeitsbestimmung. Der Ansatz arbeitet konservativ und evaluiert raum-, bild- und zeitbasierte Kohärenzen. Im Gegensatz zu PVS Verfahren beansprucht er weder eine Prekalkulation noch einen großen Speicherbedarf [15]. Die Präzision des Verfahrens kann über fünf Parameter justiert werden, um den Aufwand der Rechenleistung des jeweiligen Endgerätes anzupassen.

Das View Frustum Culling repräsentiert die erste Phase des Visibility Cullings. Es identifiziert sämtliche Animationsagenten als unsichtbar, die vollständig außerhalb der in Abbildung 6.35 dargestellten Sichtpyramide des Betrachters liegen. Gleiches gilt auch für Agenten, die sich im Raum zwischen Augpunkt und Projektionsfläche befinden. Aufgrund der durch den Raumunterteilungsbaum (siehe Abschnitt 6.3.4) beschriebenen räumlichen Kohärenzen können die folgenden Annahmen getroffen werden:

- Ist eine Zelle unsichtbar, dann sind alle Animationsagenten und Tochterzellen im Inneren der Zelle ebenfalls unsichtbar.

[15] An dieser Stelle sei daran erinnert, dass in Abschnitt 6.3.4 eine Lösung für die konsistente Pflege dynamischer Raumunterteilungsbäume beschrieben ist.

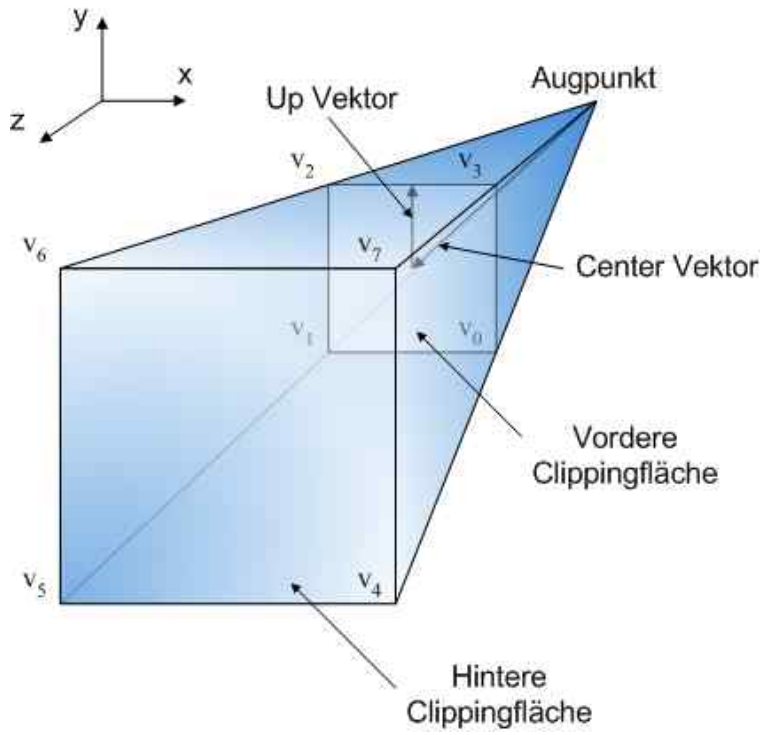

Abbildung 6.35: *Ähnlich den visuellen Informationen eines Animationsagenten ist die Sichtpyramide ebenfalls als eine Menge von Polygonen beschrieben. Die vordere Clippingfläche entspricht der Projektionsfläche. Während der Center Vektor das Zentrum der Projektionsfläche markiert, spezifiziert der Up Vektor die Orientierung des Betrachters. Die Länge des Center Vektors ist gleichbedeutend mit der Brennweite der Projektion.*

- Ist eine Zelle vollständig sichtbar, dann sind alle Animationsagenten und Tochterzellen im Inneren der Zelle ebenfalls vollständig sichtbar.

- Ist die Projektionsfläche des Betrachters vollständig oder teilweise im Inneren einer Zelle, dann sind alle Animationsagenten und Tochterzellen innerhalb der Zelle vollständig sichtbar.

- Ist die Projektionsfläche des Betrachters vollständig außerhalb einer Zelle und selbige zumindest teilweise sichtbar, dann sind alle Animationsagenten und Tochterzellen innerhalb der Zelle potentiell sichtbar.

Aus diesen Annahmen ergeben sich in Bezug auf das View Frustum Culling vier mögliche Sichtbarkeitszustände für die AABB einer Zelle beziehungsweise eines Animationsagenten:

- Liegen alle Scheitelpunkte der AABB im Inneren der Sichtpyramide, dann gilt die AABB als *Sichtbar*.

- Liegt die Projektionsfläche des Betrachters vollständig oder teilweise innerhalb der AABB, dann gilt die AABB als *Intern*.

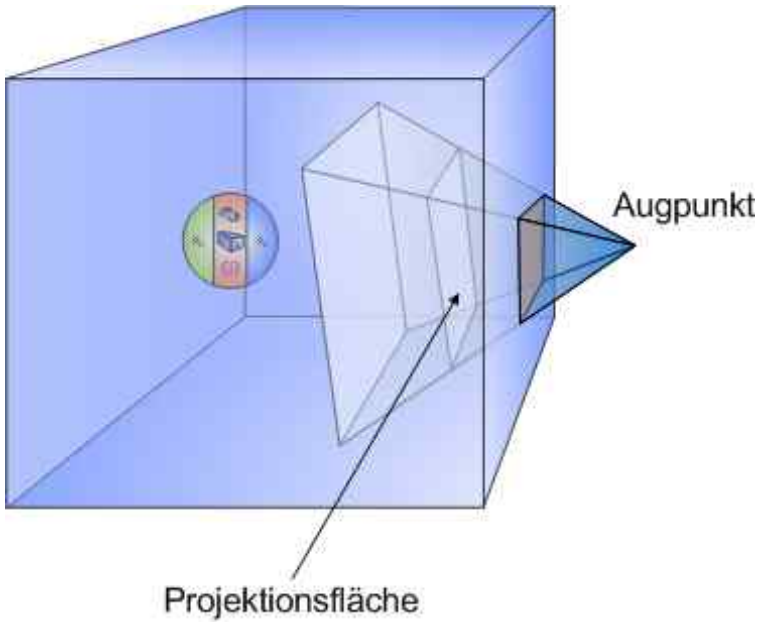

Abbildung 6.36: *Selbst wenn alle sechs Flächen einer AABB auf Sichtbarkeit getestet werden, kann es zu möglichen Fehlern kommen. In diesem Beispiel liegen sämtliche Flächen einer Zelle außerhalb des Sichtbereiches des Betrachters, die Projektionsfläche aber innerhalb der Zelle. Obgleich also die Zelle als unsichtbar identifiziert würde, könnten dortige Animationsagenten sichtbar sein. Ein Überprüfung des Zustands Intern verhindert diesen Fehler.*

- Liegt die Projektionsfläche des Betrachters vollständig außerhalb der AABB und ist diese zumindest teilweise innerhalb der Sichtpyramide, dann gilt die AABB als *Partiell Sichtbar*.

- Liegt die Projektionsfläche des Betrachters vollständig außerhalb der AABB und ist diese vollständig außerhalb der Sichtpyramide unsichtbar, dann gilt die AABB als *Unsichtbar*.

Mit dem Status *Intern* ist hier ein vierter Sichtbarkeitszustand gegeben, wohingegen andere Verfahren wie etwa [BHS98] lediglich drei Zustände spezifizieren. Der Grund für die Einführung eines weiteren Zustands resultiert aus der Frage, wie denn nun die Sichtbarkeit einer AABB zu überprüfen ist. Die zur Verfügung stehenden Informationen einer AABB bestehen aus ihren Scheitelpunkten und Flächen. Die logische Konsequenz wäre daher, die sechs Flächen der AABB auf ihre Sichtbarkeit zu testen und im Falle mindestens einer positiven Antwort auf die Sichtbarkeit der gesamten AABB zurückzuschließen. Abbildung 6.36 zeigt aber eine Situation, in der diese Vorgehensweise fehlschlägt. Wohingegen sich hier alle Flächen der AABB außerhalb des gültigen Sichtbereiches der Pyramide befinden, liegt die Projektionsfläche innerhalb der AABB. Aus diesem Grund können dort vorhandene Animationsagenten durchaus sichtbar sein [16].

Durch den Zustand *Intern* werden derartige Fehlerquellen vermieden, wobei allerdings eine exakte Identifikation dieses Status aufwändig erscheint. Ein ersterer einfacher Test könnte

[16] Bartz et al. [BMH99] gehen zwar auf derartige Situationen ein, beschreiben aber keine genaue Lösung.

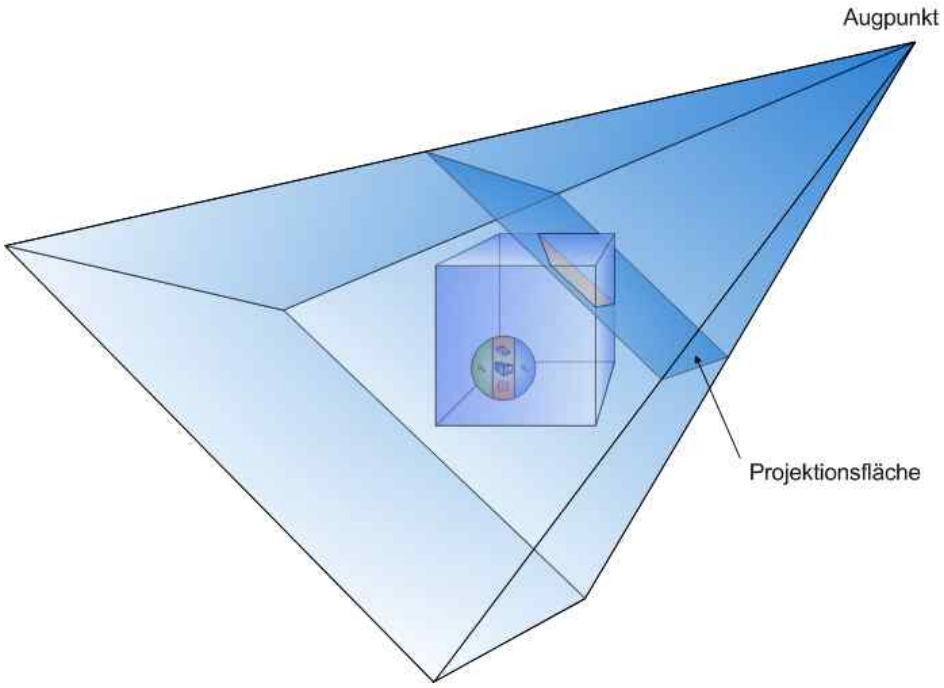

Abbildung 6.37: *Obgleich kein Scheitelpunkt der Projektionsfläche im Inneren der Zelle liegt, schneidet die Projektionsfläche die Zelle. In diesem Fall sind alle Tochterzellen beziehungsweise Animationsagenten innerhalb der Zelle potentiell sichtbar, da die vor der Projektionsfläche liegenden Elemente durchaus noch abgebildet werden können.*

daher sein, die Scheitelpunkte der Projektionsfläche auf ihre Lage bezüglich einer AABB zu analysieren. Liegt mindestens einer der Scheitelpunkte innerhalb der AABB, so ist diese AABB *Intern*. Abbildung 6.37 stellt aber einen Fall dar, in dem es zu einem fehlerhaften Ergebnis kommt.

Obgleich mit der soeben erwähnten Überprüfung der Scheitelpunkte ein großer Teil der möglichen Situationen abgedeckt ist, bleibt eine gewisse Unsicherheit. Aus diesem Grund besteht keine andere Alternative, als einen exakten Test der Projektionsfläche gegen die AABB vorzunehmen. Ein Option hierfür bestünde etwa im Clipping Algorithmus von Sutherland et al. [SH74], der jedoch in Software zu implementieren und somit der Performanz des Verfahrens abträglich wäre. Es gibt aber auch eine schnelle hardwarebasierte Lösung, die in Abschnitt 7.11.2 beschrieben wird. Für das Konzept des Verfahrens reicht es aus, dass der Zustand *Intern* existiert und effizient identifiziert werden kann. Der mit diesem Zustand betriebene Aufwand hat übrigens einen angenehmen Nebeneffekt. Wie in Abbildung 6.38 dargestellt, garantiert er nämlich, dass, wenn eine AABB den Test übersteht, ausschließlich ihre dem Betrachter zugewandten Flächen sichtbar sein können. Auf diese Weise müssen bei folgenden Tests anstelle von sechs maximal nur drei Flächen einer AABB berücksichtigt werden. Um welche Flächen es sich dabei handelt, ist infolge der achsenparallelen Eigenschaft einer AABB äußerst einfach zu bestimmen. Wie in Abbildung 6.39 gezeigt, ergeben sie sich direkt aus der Lage des Augpunktes zur AABB.

Mit jeder Hauptachse des Koordinatensystems sind jeweils diejenigen beiden Flächen einer AABB assoziiert, welche die Extrema in eine der drei möglichen Richtungen repräsentieren.

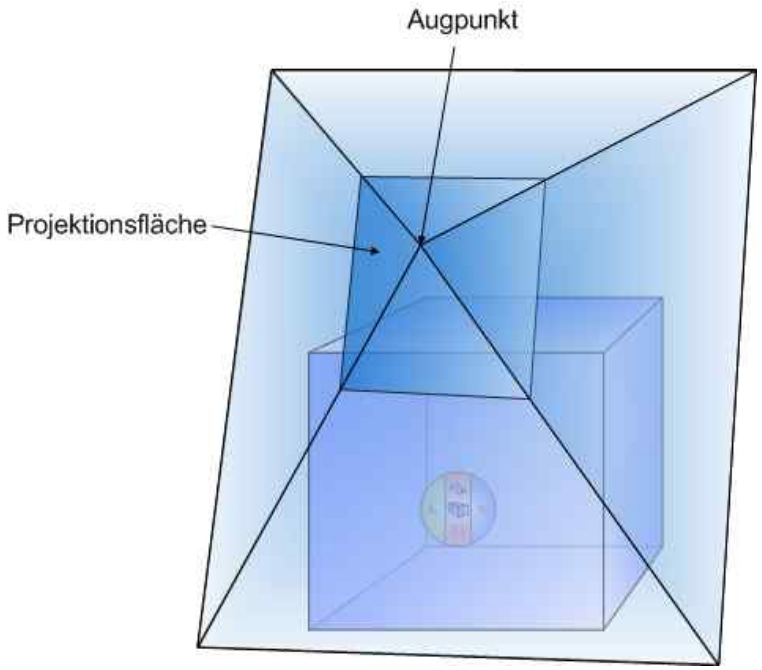

Abbildung 6.38: *Durch die Überprüfung des Zustands Intern ist sichergestellt, dass im Falle einer partiell oder vollständig sichtbaren AABB nur deren dem Betrachter zugewandten Flächen zu testen sind, wodurch sich der Aufwand deutlich reduziert.*

In Abbildung 6.39 handelt es sich dabei um die gelb gefärbten Seitenflächen einer Zelle hinsichtlich der x-Richtung. Ist die x-Komponente des Augpunktes wie im Bild I.) kleiner als der minimale x-Wert der Zelle, so ist die minimale x-Fläche der Zelle potentiell sichtbar. Ist die x-Komponente des Augpunktes dagegen wie im Bild III.) größer als der maximale x-Wert der Zelle, so ist die maximale x-Fläche der Zelle potentiell sichtbar. Liegt der Augpunkt entsprechend Bild II.) zwischen den beiden Extrema, dann sind beide Flächen definitiv nicht sichtbar. Weil also pro Dimension höchstens eine Fläche sichtbar sein kann, ergibt sich beim Erweitern auf die Dimensionen y und z übertragen die erwähnte maximale Zahl von drei sichtbaren Dreiecken.

Da die Sichtpyramide nach Abbildung 6.35 lediglich über neun Scheitelpunkte verfügt, ist die dynamische Bestimmung einer AABB der Sichtpyramide mit geringem Aufwand möglich. Um nun im Rahmen eines View Frustum Cullings die AABB A einer Zelle oder eines Animationsagenten unter Berücksichtigung der vier genannten Sichtbarkeitszustände einzuordnen, sind die folgenden Tests in der angegebenen Reihenfolge erforderlich:

1. Liegt A außerhalb der AABB der Sichtpyramide, so erhält A den Zustand *Unsichtbar*.

2. Liegen alle acht Scheitelpunkte von A innerhalb der Sichtpyramide, so ergibt sich für A der Zustand *Vollständig Sichtbar*.

3. Liegt mindestens einer der vier Scheitelpunkte der Projektionsfläche innerhalb von A, so wird A der Zustand *Intern* zugewiesen.

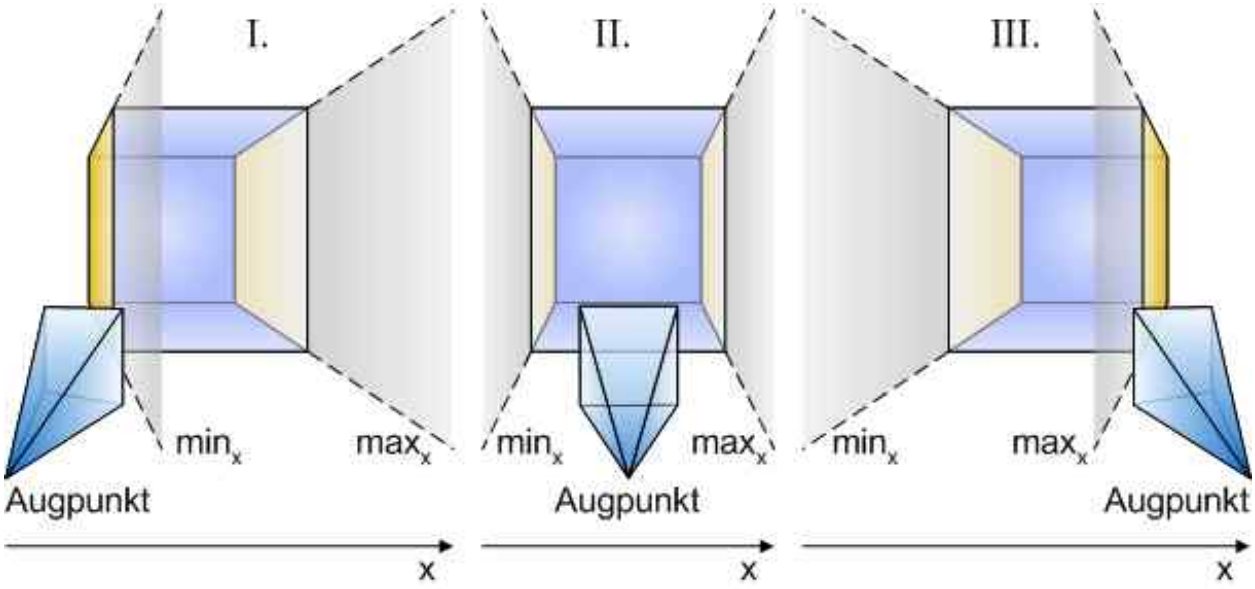

Abbildung 6.39: *Die potentiell sichtbaren Flächen einer AABB können einfach bestimmt werden, in dem der Augpunkt in Relation zu den Ebenen gesetzt werden, welche die Flächen der AABB aufspannen.*

4. Bleibt nach dem Clipping der Projektionsfläche gegen A zumindest eine Teilfläche der Projektionsfläche übrig, so gilt A ebenfalls als *Intern*. Dieser Schritt kann eventuell durch die Bestimmung der AABB der Projektionsfläche vermieden werden. Liegt A außerhalb der AABB der Projektionsfläche, so ist der exakte Test überflüssig.

5. Bleibt nach dem Clipping der potentiell sichtbaren Flächen von A gegen die Sichtpyramide zumindest eine Teilfläche übrig, so bedeutet dies für A den Zustand *Partiell Sichtbar*.

6. Der AABB wird der Zustand *Unsichtbar* zugeordnet.

Ausgehend von diesem Test und den oben genannten Annahmen hinsichtlich der räumlichen Kohärenzen des Raumunterteilungsbaumes kann eine rekursive Vorgehensweise abgeleitet werden. Dabei wird der Raumunterteilungsbaum beginnend mit Wurzel traversiert und für jede besuchte Zelle der eben erläuterte Test angewendet. Ist die Zelle unsichtbar oder vollständig sichtbar, so endet die Rekursion an dieser Stelle aufgrund der ersten beziehungsweise der zweiten Annahme. Resultiert der Test dagegen in den Zuständen *Intern* oder *Partiell Sichtbar*, so ist keine genaue Angabe über den Inhalt der Zelle möglich. Hier fällt daher zunächst der Test der in der Zelle direkt gespeicherten Animationsagenten an und danach der rekursive Test der Tochterzellen.

Das soeben erläuterte View Frustum Culling dient als eine Art Filter für das nun folgende Occlusion Culling. Insbesondere während des View Frustum Cullings als unsichtbar identifizierte Zellknoten oder Animationsagenten müssen nicht weiter berücksichtigt werden. Allerdings spielen auch die anderen ermittelten Zustände eine wichtige Rolle. Wie zuvor angedeutet basiert das Occlusion Culling auf einer Auswahl geeigneter Occluder. Dazu bieten sich große, dem Betrachter nahe gelegene Primitive an, weil hier eine hohe Wahrscheinlichkeit besteht,

dass diese Primitive einen beachtlichen Teil der Szene verdecken. Die von Coorg et al. [CT97] eingeführte Formel 4.9 berücksichtigt entsprechende Eigenschaften, hat aber den Nachteil, äußerst unpräzise zu sein. Beispielsweise kann ein Polygon als Occluder identifiziert werden, obgleich es sich außerhalb des Sichtbereiches befindet. Grundsätzlich sollte dieser Fall mit Hilfe des View Frustum Cullings vermieden werden, was aber nur eingeschränkt möglich ist. So beansprucht ein als partiell sichtbar erkannter Animationsagent eventuell einen Großteil des Sichtbereiches, beinhaltet jedoch unsichtbare Primitive. Die Genauigkeit des View Frustum Cullings reicht also nur bis zur Ebene der Animationsagenten aber nicht bis zu den Primitiven der Agenten.

Wesentlich exakter als Gleichung 4.9 ist die Bestimmung der wirklich von einem Primitiv beziehungsweise Polygon gerenderten Pixel im Bildbereich. Im folgenden wird die Zahl dieser Pixel als *Gerenderte Pixelmenge* (GPM) bezeichnet. Die NVIDIA-Occlusion-Query liefert die GPM, gehört aber leider nicht zum Standard derzeitiger Grafikbibliotheken. Mit Hilfe der Formel

$$2A(P) = \sum_{i=0}^{n-1}(x_i + x_{i+1})(y_{i+1} - y_i) \tag{6.6}$$

ist allerdings ein sehr gute Annäherung der gerenderten Pixelmenge möglich. Gleichung 6.6 berechnet den Flächeninhalt eines Polygons P über dessen auf die Projektionsebene des Betrachters projizierten 2D Scheitelpunkte. Somit muss ein Polygon also vor der Anwendung der Gleichung gegen die Sichtpyramide geclippt und anschließend der perspektivischen Transformation unterworfen werden. In einer reinen Software Lösung bedeutet dies einen sehr aufwändigen Prozess. Glücklicherweise können die benötigten Informationen direkt aus dem OpenGL Feedback Buffer gewonnen werden, worauf Abschnitt 7.11.2 eingeht. Hier sind erst einmal einige Annahmen von Bedeutung, welche die Zahl der zu überprüfenden Polygone stark einschränken.

- Ist eine Zelle des Raumunterteilungsbaumes unsichtbar, dann können weder die Zelle selbst noch ihre Tochterzellen Animationagenten mit Occludern beinhalten.
- Ist eine Zelle sichtbar oder partiell sichtbar und liegt die GPM ihrer AABB unterhalb einem benutzerdefinierten Grenzwert T_1, dann kann weder diese Zelle noch eine ihrer Tochterzellen Animationsagenten mit Occludern beinhalten.
- Ist ein Animationsagent unsichtbar, dann kann er keine Occluder beinhalten.
- Ist ein Animationsagent sichtbar oder partiell sichtbar und liegt die GPM seiner AABB unterhalb einem benutzerdefinierten Grenzwert T_2, dann kann dieser Animationsagent keine Occluder beinhalten.
- Ist ein Animationsagent sichtbar und liegt die GPM seiner AABB geteilt durch die Anzahl seiner Polygone unterhalb einem benutzerdefinierten Grenzwert T_3, dann kann dieser Animationsagent keine Occluder beinhalten.

Wird die GPM einer AABB berechnet, dann kommen weitere Optimierungen zum Tragen. Ist die AABB partiell sichtbar, dann müssen lediglich die dem Betrachter zugewandten Flächen der AABB berücksichtigt werden. Ist die AABB sichtbar, dann kann mit Hilfe einer *Lookup Table* die konvexe Hülle der 2D Projektion der AABB sehr schnell bestimmt werden. Ausgehend von dem zuvor erläuterten Ansatz zur Identifikation der dem Betrachter zugewandten Flächen gibt es aufgrund der drei Dimensionen x, y und z insgesamt $3^3 = 27$ verschiedene Positionen, an denen sich der Betrachter in Relation zu einer AABB befinden kann. Entsprechend beinhaltet die Lookup Table 27 Einträge. Mit Hilfe der Lookup Table ist lediglich die Berechnung der GPM der konvexen Hülle erforderlich und nicht die der maximal drei Frontflächen. Die folgende Rekursion dient der Selektion der Occluder in jedem Frame und traversiert hierzu den Raumunterteilungsbaum.

1. Besitzt die aktuelle Zelle den Zustand *Intern*, dann werden für jeden direkt in der Zelle eingetragenen Animationsagenten die Schritte a-c durchgeführt. Anschließend fährt die Rekursion mit den Tochterzellen der Zelle fort. Dabei sollten die Tochterzellen in einer Front-to-Back Reihenfolge bearbeitet werden.

   (a) Ist der Animationsagent intern, dann wird für jedes seiner Polygone die GPM berechnet. Überschreitet die GPM eines Polygons einen benutzerdefinierten Grenzwert T_4, dann ist dieses Polygon ein Occluder. Überschreitet die Anzahl der gefundenen Occluder einen benutzerdefinierten Grenzwert T_5, so ist die Occluder Selektion beendet.

   (b) Ist der Animationsagent partiell sichtbar, dann wird die GPM seiner AABB berechnet. Überschreitet die GPM den Grenzwert T_2, dann werden wie in Schritt a) die Polygone des Agenten getestet.

   (c) Ist der Animationsagent sichtbar, dann wird die GPM der konvexen Hülle seiner AABB ermittelt. Überschreitet die GPM den Grenzwert T_2, dann wird selbige durch die Anzahl der Polygone des Animationsagenten geteilt. Überschreitet das Ergebnis den Grenzwert T_3, dann werden wie in Schritt a) die Polygone des Agenten überprüft.

2. Ist die aktuelle Zelle teilweise sichtbar, dann wird die GPM ihrer AABB berechnet. Übersteigt die GPM den Grenzwert T_1, dann werden für jeden direkt in der Zelle eingetragenen Animationsagenten die Schritte a-c durchgeführt. Anschließend fährt die Rekursion wie in Schritt 1) mit den Tochterzellen fort.

3. Ist die aktuelle Zelle sichtbar, dann wird die GPM der konvexen Hülle ihrer AABB betimmt. Überschreitet die GPM den Grenzwert T_1, dann werden für jeden direkt in der Zelle eingetragenen Animationsagenten die Schritte a-c durchgeführt. Anschließend fährt die Rekursion wie in Schritt 1) mit den Tochterzellen fort.

Entsprechend zu den benutzerdefinierten Grenzwerten existieren fünf Parameter, welche die Occluder Selektion kontrollieren. Sind diese Parameter geeignet ausgewählt, dann überprüft der Ansatz während der Selektion nur einige wenige Animationsagenten auf Occluder. Da die berechneten GPMs durch die Anzahl der Pixel des Gesamtbildschirms geteilt werden, sind alle Parameter in Prozent definiert und somit für beliebige Auflösungen gültig.

- Parameter P_1 definiert die minimale GPM der AABB einer Zelle, damit letztere Occluder beinhalten kann.
- Parameter P_2 definiert die minimale GPM der AABB eines Animationsagenten, damit letzterer Occluder beinhalten kann.
- Parameter P_3 definiert die minimale durchschnittliche GPM der AABB eines sichtbaren Animationsagenten, damit letzterer Occluder beinhalten kann.
- Parameter P_4 definiert die minimale GPM eines Polygons, damit selbiges als Occluder geeignet ist.
- Parameter P_5 definiert die maximale Anzahl an Occludern.

Da die vorgestellte Occluder Selektion innerhalb jeden Frames den Raumunterteilungsbaum auf potentielle Occluder überprüft, sind Veränderungen des Baumes von Frame zu Frame für das Verfahren transparent. Aus diesem Grund können auch dynamische Szenen verarbeitet werden. Üblicherweise sind die Transformationen der aktiven Animationsagenten und der Sichtpyramide des Betrachters konsistent, d.h. die visuellen Unterschiede zwischen zwei Frames fallen gering aus. Somit sollten die für einen Frame F_1 gewählten Occluder auch für einen Frame F_2 geeignet sein. Um die temporären Kohärenzen zwischen zwei Frames auszunutzen, werden deshalb zunächst die Occluder aus dem vorhergehenden Frame im aktuellen Frame auf ihre erneute Tauglichkeit getestet.

Bevor nun der eigentliche Sichtbarkeitstest erfolgt, müssen die identifizierten Occluder in einer Aufbereitungsphase bearbeitet werden. Die Vorgehensweise orientiert sich dabei an dem Verfahren von Bittner et al. [BHS98], bietet aber einige Verbesserungen. Abbildung 6.40 illustriert die grundlegende Vorgehensweise. Jede Kante eines Occluder spannt zusammen mit dem Augpunkt des Betrachters eine Schattenebene auf, wobei alle Schattenebenen des Occluders ein Volumen bilden. Innerhalb dieses Volumens ist der Bereich vor dem Occluder für den Betrachter sichtbar und die Region hinter dem Occluder unsichtbar. Das Ziel ist somit, diejenigen Animationsagenten einer Szene zu identifizieren, welche vollständig im unsichtbaren Schattenbereich eines Occluders liegen und daher bei einer Ausgabe nicht berücksichtigt werden müssen. Bei einer großen Anzahl von Occludern entspricht dies einem sehr aufwändigen Unterfangen. Aus diesem Grund sortieren Bittner et al. die Occluder in einen sogenannten *Occlusion Tree* beziehungsweise *Verdeckungsbaum* ein. Dieser Verdeckunsgbaum ist vom Aufbau äquivalent zu dem bereits in Abschnitt 6.5.2 vorgestellten Clientbaum. Der Unterschied besteht darin, dass jeder Zonenknoten innerhalb des Verdeckungsbaumes eine Schattenebene repräsentiert und jeder Clientknoten anstelle eines Clients den eigentlichen Occluder. Entsprechend handelt es sich hier um *Schattenknoten* beziehungsweise um *Occluderknoten*. Da ein Occluder sich im Schattenvolumen eines anderen Occluders befinden kann, müssen die Occluder vor dem Einfügen in den Verdeckungsbaum in einer Front-to-Back Reihenfolge sortiert werden. Dies kann etwa mit Hilfe eines BSP Baumes nach Schumacker et al. [SBGS69] geschehen. Aufgrund der dabei eventuell auftretenden Unterteilungen der Occluder ist es möglich, dass sich die Anzahl der Occluder vergrößert. Gleichzeitig verringert sich die GPM der unterteilten Occluder und somit auch ihre Effizienz bezüglich des Occlusion Cullings.

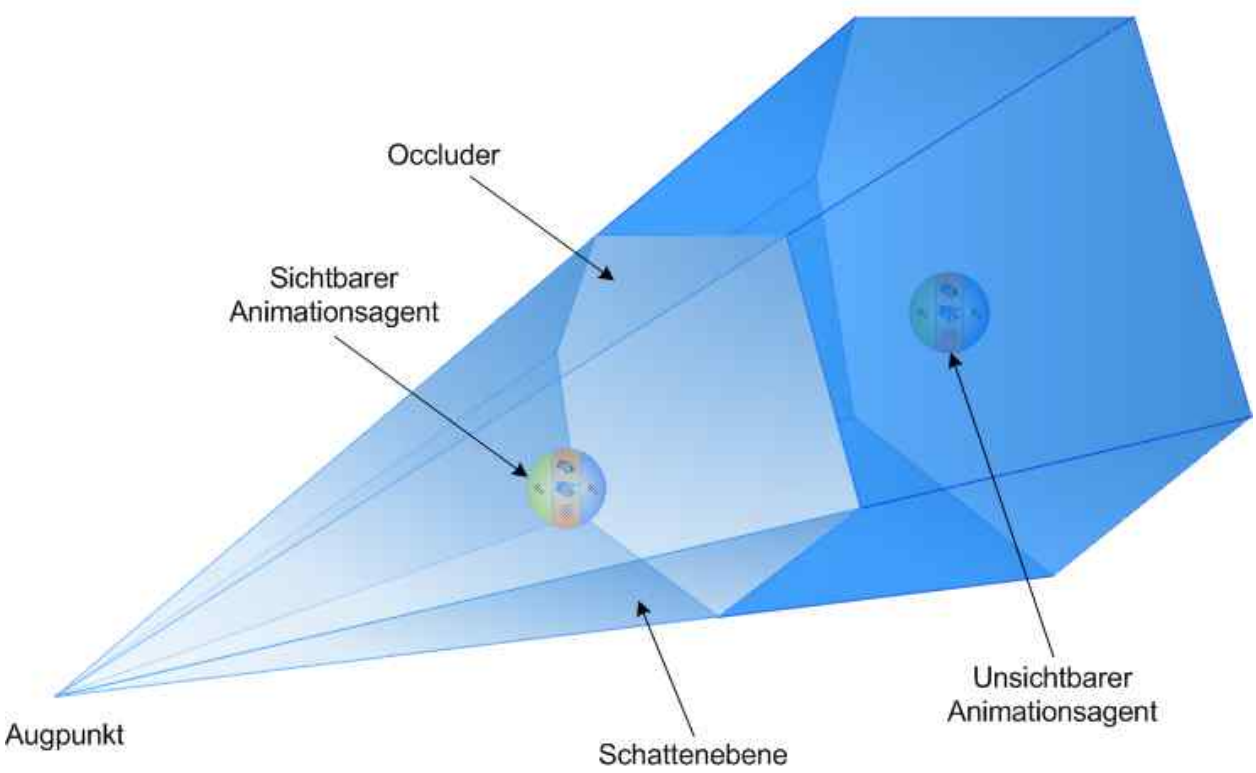

Abbildung 6.40: *Jede Kante eines Occluders spannt zusammen mit dem Augpunkt des Betrachters eine Schattenebene auf. Das hinter dem Occluder durch die Schattenebenen eingeschlossene Volumen ist für den Betrachter unsichtbar.*

Abbildung 6.41 beschreibt eine Möglichkeit, wie die Anzahl der Unterteilungen reduziert werden kann.

Hierzu wird der Bildbereich nach dem Prinzip eines Quadtrees in Zellen unterteilt und jeder selektierte Occluder in diejenige Zelle eingefügt, welche den Occluder vollständig umschließt. Während der Front-to-Back Sortierung müssen die Occluder, die sich in unterschiedlichen Pfaden des Quadtrees befinden, nicht gegeneinander getestet werden. Auf diese Weise sind bildbasierte Kohärenzen der Occluder berücksichtigt.

Das Einfügen der Occluder in den Occlusion Tree ist identisch zu der in Abschnitt 6.5.2 beschriebenen Vorgehensweise. Im Gegensatz zu Bittner et al. erfolgt die Einsortierung allerdings nicht über die dreidimensionalen Schattenebenen, sondern mittels der zweidimensionalen Projektion eines Occluders auf den Bildbereich. Somit repräsentiert ein Schattenknoten lediglich eine Kante des Occluders. Jede Kante unterteilt den zweidimensionalen Bildbereich in zwei Halbräume, wobei über die Orientierung des Occluders festgestellt werden kann, welcher Halbraum dem inneren beziehungsweise äußeren Bereich des Occluders zuzuordnen ist. Abbildung 6.42 illustriert den Aufbau des Verdeckungsbaumes nach dem Einfügen zweier Occluder A und B.

Per Definition liegen die Kanten eines Occluders stets im inneren Halbraum, der durch eine andere Kante des gleichen Occluders aufgespannt wird. Aus diesem Grund ergeben sich

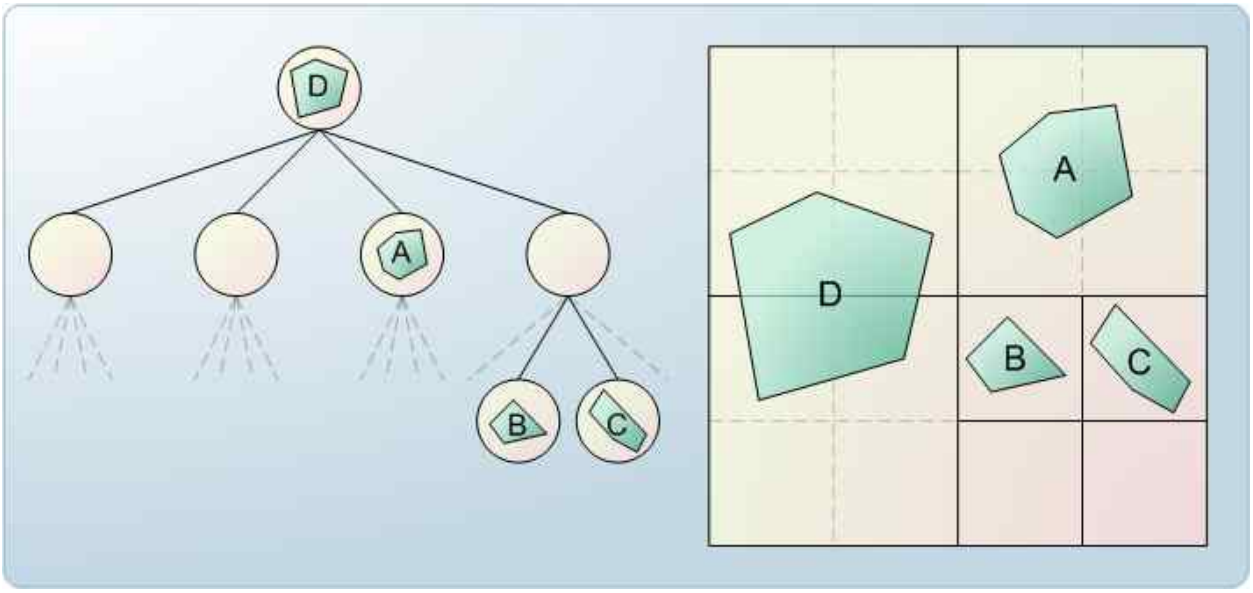

Abbildung 6.41: *Der Bildbereich wird in einen Quadtree unterteilt und die Occluder derjenigen Zelle zugeordnet, die sie vollständig umschließt. Occluder, die ausgehend von der Wurzel in unterschiedlichen Pfaden liegen, müssen während der Front-to-Back Sortierung mittels eines BSP Baumes nicht gegeneinander getestet werden.*

die schon in Abschnitt 6.5.2 angesprochenen Knotenketten. Der Occluder B liegt sowohl im inneren als auch im äußeren Halbraum der Kante b_A des Occluders A. Der Occluder B muss also gegen b_A geclippt und die beiden daraus resultierenden Teiloccluder getrennt durch den Verdeckungsbaum propagiert werden. Der äußere Teiloccluder wird analog zu A als Knotenkette an b_A angefügt. Der innere Teiloccluder erreicht den Occluderknoten von A. Aufgrund der Front-to-Back Sortierung muss B hinter A liegen. Dieser Teiloccluder würde somit lediglich Animationsagenten der Szene verdecken, die sich ohnehin schon im Schatten von A befinden. Daher kann der innere Teiloccluder von B verworfen werden. Nach dem Einfügen aller Occluder repräsentiert der Verdeckungsbaum exakt die Bildbereiche, welche durch die Occluder belegt werden. Das Konzept des Verdeckungsbaumes entspricht also letztlich einem Occluder Fusion Ansatz (siehe Abschnitt 4.6).

Jeder verbleibende Occluder ist innerhalb des Verdeckungsbaumes durch einen Occluderknoten repräsentiert. Dort sind die Parameter der Ebenengleichung des Occluders eingetragen. Zum Berechnen der Ebenengleichung werden spezielle Scheitelpunkte verwendet. Die x- und y-Komponente dieser Scheitelpunkte entsprechen der x- und y-Komponente der zugehörigen Scheitelpunkte der 2D Projektion. Die z-Komponente ist ein Wert im Bereich $0 \leq z \leq 1$ und beschreibt die normierte Distanz zur hinteren Clipping Ebene der Sichtpyramide. 0 bedeutet der Scheitelpunkt liegt auf der vorderen Clipping Ebene und 1 der Scheitelpunkt liegt auf der hinteren Clipping Ebene. Im folgenden werden derartige Scheitelpunkte als $2\frac{1}{2}$D Scheitelpunkte bezeichnet. Sie entsprechen den Werten, welche der OpenGL Feedback Buffer beim Rendern eines Primitives zurück liefert.

Mit Hilfe des Occlusion Trees ist eine schelle Bestimmung möglich, ob ein planares, konvexes Polygon P sichtbar, teilweise sichtbar oder unsichtbar ist. Zu diesem Zweck müssen alle Scheitelpunkte des Polygons $2\frac{1}{2}$D Scheitelpunkte sein. Das Polygon wird äquivalent zu den

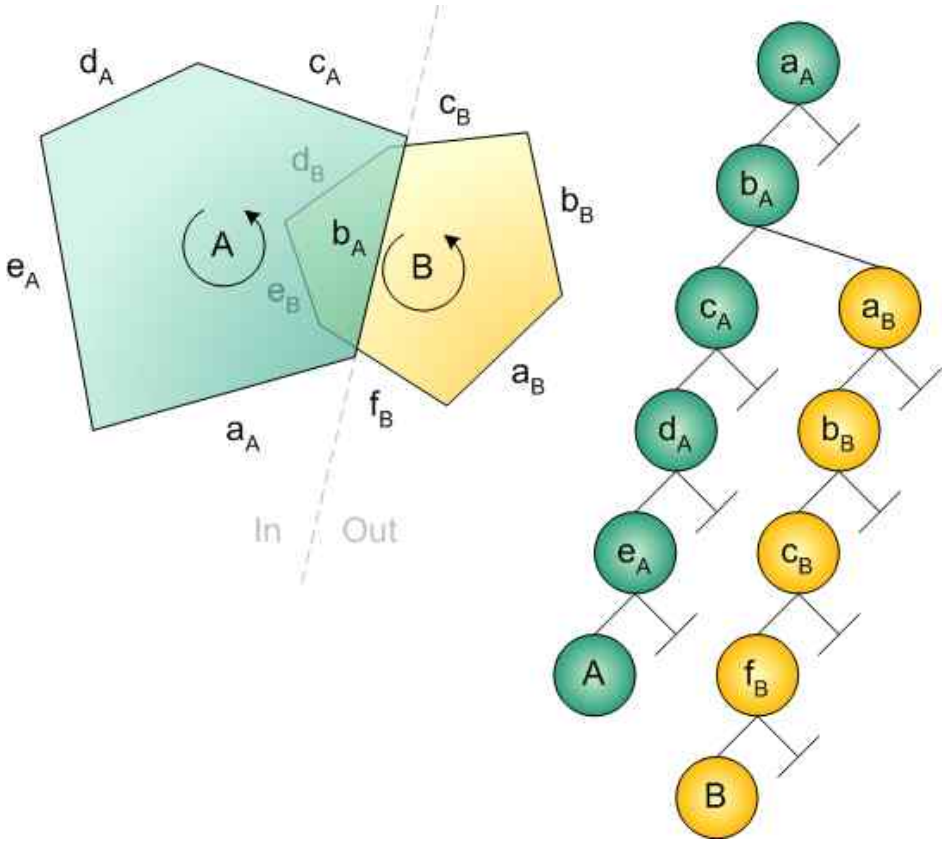

Abbildung 6.42: *Der Verdeckungsbaum nach dem Einfügen zweier Occluder A und B. Da B hinter A liegt wird der durch A verdeckte Bereich von B nicht weiter berücksichtigt.*

AABBs in Abschnitt 6.5.3 durch den Occlusion Tree propagiert. Liegt das Polygon P vollständig im inneren Halbraum eines Schattenknotens, so wird es durch den linken Teilbaum propagiert. Befindet sich das Polygon P komplett im äußeren Halbraum, dann erfolgt eine Traversierung des rechten Teilbaumes. Andernfalls ist ein Schneiden des Polygons P gegen die Kante des Schattenknotens erforderlich. Hierzu werden die x- und die y- Komponente der Schnittpunkte analog zu einer Sutherland-Hodgman Berechnung [SH74] im 2D Raum ermittelt. Die z-Komponente wird ausgehend vom Startpunkt der Schnittkante proportional durch den Abstand bestimmt. Anschließend werden die beiden resultierenden Polygone P' und P'' rekursiv durch die jeweiligen Teilbäume propagiert. Sobald ein Polygon einen Occluderknoten betritt, sind die Scheitelpunkte des Polygons in die Ebenengleichung des Occluderknotens einzusetzen. Liegen alle Scheitelpunkte hinter dem Occluder und somit auf der dem Betrachter abgewandten Seite, so ist das Polygon unsichtbar. Befinden sich alle Scheitelpunkte vor dem Occluder, dann gilt das Polygon als vollständig sichtbar. Ansonsten ist das Polygon partiell sichtbar. Ein Polygon ist ebenfalls vollständig sichtbar, sofern es dem äußeren Halbraum eines Schattenknotens zugeordnet wird und dort kein weiterer Teilbaum vorhanden ist.

Aufgrund der möglich Aufspaltung des Polygons P während der Traversierung des Verdeckungsbaumes müssen die Ergebnisse der einzelnen Teilpolygone kombiniert werden. Sind alle Teilpolygone vollständig sichtbar oder unsichtbar, dann gilt gleiches für das gesamte

Polygon P. In allen anderen Fällen ist P partiell sichtbar. Weil die AABB einer Zelle oder eines Animationsagenten aus konvexen Polygonen besteht, könnten diese Polygone wie im Falle von Bittner et al. einzeln mit Hilfe des Verdeckunsgbaumes auf Sichtbarkeit getestet werden. Diese Vorgehensweise ist jedoch nicht sehr effizient, weshalb jede AABB durch die Kantenmenge ihrer dem Betrachter zugewandten Flächen repräsentiert wird. Den Flächen gemeinsame Kanten sind nur jeweils einmal in diese Kantenmenge eingetragen. Sämtliche Kanten verfügen über $2\frac{1}{2}$D Scheitelpunkte. Eine Sortierung der Kanten als Linienzug ist nicht erforderlich. Es kommt nicht darauf an, ein Polygon oder gar eine AABB vollständig durch den Verdeckungsbaum zu traversieren. Entscheidend ist vielmehr, in welchem Bezug die Scheitelpunkte eines Polygons oder einer AABB zu den Occludern stehen. Da während der Traversierung des Verdeckungsbaumes durch die Unterteilung eines Polygons eventuell neue Scheitelpunkte beziehungsweise Schnittpunkte zu ermitteln sind, ist zumindest die topologische Information der Kanten notwendig. Ein Kantenmenge gilt somit als vollständig sichtbar, sofern alle ihre Scheitelpunkte einschließlich möglicher Schnittpunkte entweder vor den Occludern oder außerhalb des Bildbereiches der Occluder liegen. Sind dagegen alle Scheitelpunkte beziehungsweise Schnittpunkte unsichtbar, dann trifft gleiches auch auf die Kantenmenge zu. Ansonsten ist die Kantenmenge als partiell sichtbar zu werten.

Die folgenden Schritte repräsentieren den eigentlichen rekursiven Sichtbarkeitstest und propagieren ausgehend von der Wurzel des Raumunterteilungsbaumes die Kantenmenge einer AABB durch den Verdeckungsbaum. Dabei sind nur während des View Frustum Cullings als intern, sichtbar oder partiell sichtbar identifizierte Zellen beziehungsweise Animationsagenten zu berücksichtigen.

1. Wurde die aktuelle Zelle während des View Frustum Cullings als intern deklariert, so gilt die Zelle automatisch als potentiell sichtbar. Aus diesem Grund werden für jeden in der Zelle eingetragenen Animationsagenten die Schritte a-b durchgeführt. Anschließend fährt die Rekursion mit den Tochterzellen der Zelle über die Schritte 1-2 fort.

   (a) Wurde ein Animationsagent während des View Frustum Cullings als sichtbar oder partiell sichtbar identifiziert, dann wird die $2\frac{1}{2}$D Kantenmenge seiner AABB durch den Verdeckungsbaum propagiert. Je nach Resultat ist der Animationsagent bei einer Visualisierung zu berücksichtigen oder zu vernachlässigen.

   (b) Wurde ein Animationsagent während des View Frustum Cullings als intern deklariert, so ist der Animationsagent in jedem Fall zu berücksichtigen.

2. Wurde die aktuelle Zelle während des View Frustum Cullings als sichtbar oder partiell sichtbar deklariert, dann wird die $2\frac{1}{2}$D Kantenmenge ihrer AABB durch den Verdeckungsbaum propagiert. Ist die Kantenmenge vollständig sichtbar, dann kann die Rekursion an dieser Stelle beendet werden, weil selbiges auch auf alle Animationsagenten und Tochterzellen innerhalb der Zelle zutrifft. Sämtliche betroffenen Animationsagenten sind bei einer Ausgabe zu berücksichtigen. Ist die Kantenmenge dagegen teilweise sichtbar, so ist die gleiche Vorgehensweise wie in Schritt 1) notwendig.

Abbildung 6.43: *Für die Navigation wird die Bildfäche in mehrere Felder unterteilt, welche jeweils mit einer bestimmten Form der Navigation korrespondieren. Die mit dem Stift oder der Maus ausgewählte Position innerhalb eines Feldes hat Auswirkungen auf die Geschwindigkeit der resultierenden Bewegung. Das Informationsfeld rechts unten zeigt Geschwindigkeit und Bewegungsform an.*

6.7.3 Eine allgemeine Navigationsschnittstelle

Mobile Endgeräte wie etwa PDAs weisen im Vergleich zu einem Standard PCs keine entsprechende Vielzahl an Interaktionsmöglichkeiten auf. Beispielsweise fehlt dort die Option einer Tastatur- oder Mauseingabe. Das Problem im Umgang mit mobilen Geräten liegt oftmals nicht alleine in den vermeintlich eingeschränkten Interaktionsformen. Vielmehr spielt auch die Übung mit dem Endgerät eine entscheidende Rolle. Einem PC Benutzer, der über Jahre hinweg die Eingabe per Maus und Tastatur perfektioniert hat, fällt es schwer, auf einmal Buchstaben per PDA Stift einzugeben. Aus diesem Grund wird in der vorliegenden Arbeit für die Navigation innerhalb einer 3D Szene ein Interaktionsmodell verwendet, das sowohl auf PCs als auch auf mobilen Endgeräten zum Einsatz kommen kann. Der Benutzer erhält somit die Gelegenheit, die auf dem PCs gewonnen Erfahrungen ebenso auf dem PDA einzusetzen.

Wie in Abschnitt 5.1.2 angedeutet, existiert mit Ultima Underworld bereits seit mehreren Jahren eine Applikation, die eine für den PDA taugliche Navigationsschnittstelle anbietet. Nach Abbildung 6.43 wird der Bildbereich dabei in mehrere Felder unterteilt. Jedes Feld repräsentiert eine bestimmte Art der Navigation. Während die geraden Pfeile eine geradlinige Translation symbolisieren, bedeuten die runden Pfeile eine Drehung des Benutzers. Das Feld in der Mitte der geraden Pfeile entspricht einer neutralen Zone, d.h. hier erfolgen keine Reaktionen auf eine Eingabe. Gerade Pfeile, die nach oben zeigen, implizieren eine Vorwärtsbewegung in die Blickrichtung des Betrachters. Felder mit diagonalen Pfeilen ergänzen die entsprechende Navigation um eine Seitwärtsbewegung. Analog symbolisieren Felder mit geraden, abwärts gerichteten Pfeilen eine Rückwärtsbewegung. Seitliche, gerade Pfeile stellen eine reine Seitwärtsbewegung dar. Alle geradlinigen Translationen erfolgen parallel zu den Hauptachsen des lokalen Koordinatensystems des Betrachters. Der runde, obere Pfeil beschreibt ein Blicken nach unten, der runde, untere Falle ein Blicken nach oben. Die runden Pfeile an der Seite bedeuten eine Drehung nach links beziehungsweise rechts.

Die Navigation wird auf einem PDA dadurch ausgelöst, dass der Benutzer mit dem Stift ein Feld der Bildfläche berührt. Die Geschwindigkeit der Bewegung ist abhängig von der gewählten Stiftposition innerhalb des Feldes. Je höher beispielsweise ein Benutzer das Feld mit dem geraden Vorwärtspfeil selektiert, desto schneller wird die Vorwärtsbewegung. Im Gegensatz zu einem PC besteht auf Seiten des PDAs das Problem, dem Benutzer die Grenzen zwischen den einzelnen Feldern anzugeben. Im Falle des PCs ist dies einfach mit Hilfe des Mauspfeils möglich. Da der Mauspfeil permanent vorhanden ist und stets zu den Eingaben der Maus korrespondiert, kann das Betreten eines Feldes durch das Einblenden einer entsprechenden Pfeilform anstelle des normalen Cursors symbolisiert werden. Die Angabe der Geschwindigkeit ist über die Länge des Pfeils möglich. Hinsichtlich des PDAs gibt es die Option, einfach auf das zunehmende Gespür des Benutzers für das gewünschte Feld zu hoffen. Eine sinnvolle Alternative ist aber das optionale Einblenden von Hilfslinien wie in Abbildung 6.43 zu sehen. Zudem kann die selektierte Form der Navigation durch die Visualisierung des mit der Bewegung assoziierten Pfeils verdeutlicht werden. Abbildung 6.43 zeigt hierzu ein kleines Informationsfeld rechts unten in der Bildfläche des PDAs. Dort befindet sich auch eine Skala für die Angabe der momentanen Navigationsgeschwindigkeit.

Drehungen können analog zu den geradlinigen Navigationsformen beschrieben werden, d.h. der Benutzer berührt ein Feld und in Abhängigkeit von der selektierten Position rotiert der Benutzer mit entsprechender Geschwindigkeit in die gewählte Richtung. Mitunter ist aber die Vorgabe insbesondere von Drehgeschwindigkeiten lästig, beispielsweise wenn der Benutzer die Rückseite eines Modells betrachten möchte, die Rotation aber Grad für Grad vonstatten geht. Eine günstigere Vorgehensweise ist daher, ein Feld mit einem gekrümmten Pfeil zu selektieren und dann den Stift oder die Maus entlang der Bewegungsrichtung zu ziehen. Je schneller das Ziehen, desto schneller erfolgt die Rotation.

6.8 Die Simulation- und Animation-Komponente

Die Simulation-Komponente des Server berechnet die der Szene zugrunde liegende Simulationen und löst dabei eventuelle, von Benutzern eingehende Interaktionen auf. Eine Ma-

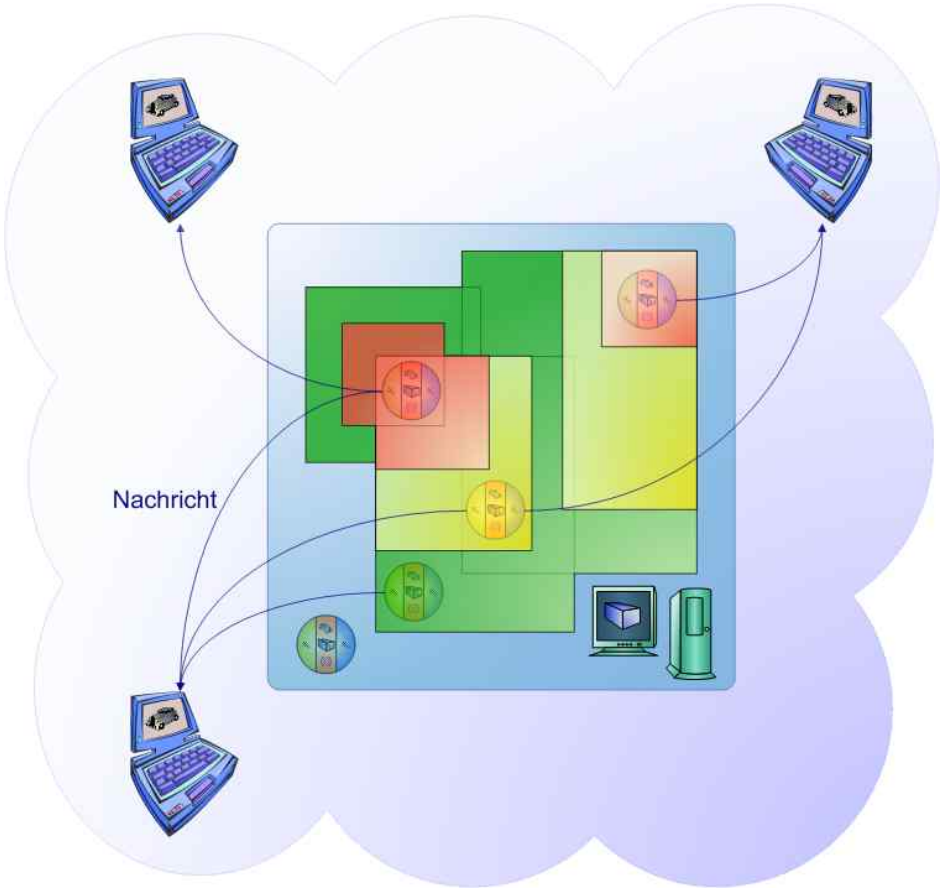

Abbildung 6.44: *Die von der Scheduler-Komponente ermittelten Prioritäten der Animationsagenten können von der Simulation-Komponente für die Reduktion des Kommunikationsaufwand eingesetzt werden, der mit der Animation und Synchronisation dynamischer Vorgänge anfällt.*

nipulation der Szene wird also erst von der Präsentation-Komponente des Verursachers an die Simulation-Komponente übermittelt. Diese Vorgehensweise hat gegenüber einem Peer-to-Peer Ansatz den Vorteil, dass die Simulation-Komponente simultane Interaktionen identifizieren und synchronisieren kann. Beispielsweise ist es möglich, dass mehrere Benutzer den gleichen Animationsagenten selektieren und transformieren möchten. Durch die zentrale Funktion der Simulation-Komponente wird die Kontrolle über den Agenten dem ersten Benutzer zugesprochen und eventuelle Nachfolger solange vertröstet, bis der Benutzer die Manipulation des Agenten beendet hat.

Eine Veränderung des Agenten wirkt sich zunächst auf die lokale Szenenpräsentation des Server aus und wird dann durch die Simulation-Komponente an die Clients weitergeleitet. Auf diese Weise ist unter Voraussetzung äquivalenter Latenzzeiten keiner der Clients benachteiligt. Während der Aktualisierung der Clients kann die Simulation-Komponente nach der Intention von Funkhouser [Fun95] den Kommunikationsaufwand durch die Berücksichtigung visueller Kriterien reduzieren. Abbildung 6.44 zeigt hierzu die Area-of-Interest dreier Clients, deren Identifikation durch die Scheduler-Komponente erfolgt. Die dort in Bezug auf die Areas-of-Interest aller Clients ermittelten Prioritäten ermöglichen eine selektive Synchronisation. Aufgrund der anfallenden Latenzzeiten beim Übersenden der Nachrichten nimmt die

Scheduler-Komponente jedoch keine den Prioritäten entsprechende Einstufung vor. Sie sendet einem Client die Aktualisierung all derjenigen Animationsagenten zu, die sich zumindest in der präventiven Zone des Clients befinden.

Neben der Berechnung und Synchronisation der dynamischen Vorgänge zeichnet sich die Simulation-Komponente auch für die Differenzierung zwischen aktiven und passiven Animationsagenten auf Seiten des Servers verantwortlich. Im Gegensatz zu den passiven Agenten, traversiert die Simulationskomponente einmal pro Zeitschritt der Simulation alle aktiven Animationsagenten über überprüft sie auf mögliche Aktionen. Von diesen Aktionen betroffene passive Animationsagenten werden dann vorübergehend in den Status aktiver Agenten erhoben.

6.9 Die Netzwerk-Komponente

Die Netzwerk-Komponente bietet für die übrigen Komponenten eine Abstraktion der Verbindung zwischen Client und Server, d.h. die Trennung durch das Netzwerk bleibt weitestgehend transparent. Die Netzwerk-Komponente wird mit dem Versenden und dem Empfang von Nachrichten beauftragt und behandelt eigenständig mögliche Fehler.

Pro Client-Server Verbindung existieren zwei Kanäle, zum einen der Kontrollkanal und zum anderen der Datenkanal. Ersterer ist für das Versenden kurzer Befehle und Kommandos angelegt, letzterer zeichnet sich für den Transfer großer Datenmengen verantwortlich. Auf diese Weise wird eine längere Blockade der Kommandos durch das Verarbeiten großer Datenblöcke vermieden. Weiterhin können unterschiedliche Netzwerk-Protokolle für die jeweiligen Kanäle zum Einsatz kommen. Beispielsweise bietet sich für den Transfer der Daten ein UDP Protokoll und für das Versenden der Befehle ein TCP/IP Protokoll an.

Die Netzwerk-Komponenten wird detaillierter von Abschnitt 7.9 behandelt, da hier lediglich gängige Vorgehensweisen verwendet werden.

6.10 Zusammenfassung

In diesem Kapitel wurde ein Konzept zur Übertragung und Visualisierung großer, dynamischer und interaktiver 3D Szenen auf verteilten Endgeräten vorgestellt. Das Konzept orientiert sich dabei an den in Abschnitt 5.1 analysierten Anforderungen.

Die grundlegende Entscheidung des Konzepts liegt in der Wahl zugunsten eines Peer-to-Peer oder eines Client-Server-Systems. Erstere erlauben eine gleichmäßigere Auslastung der Endgeräte, letztere die einfachere und effizientere Synchronisation dynamischer Vorgänge. Da sich im Falle eines Client-Server Systems die von Funkhouser vorgelegten Ergebnisse auch auf die Übertragung der visuellen Informationen einer Szene erweitern lassen und der Schwerpunkt der vorliegenden Arbeit auf der Berücksichtigung dynamischer Szenen liegt, fällt die Wahl letztlich auf einen Client-Server-Ansatz.

Abschnitt 6.1 stellt nun die Frage nach der Rollenverteilung zwischen Server und Client und versucht, jeweils anfallende Aufgaben zu identifizieren und den beiden Instanzen zuzuordnen. Während der Schwerpunkt des Servers in der Bereitstellung, Distribution sowie Synchronisation einer dynamischen Szene liegt, ist die wesentliche Herausforderung des Clients die Interaktion mit dem Benutzer, wozu sowohl die Visualisierung der empfangenen Informationen als auch die Behandlung möglicher Eingaben zu zählen ist. Anhand dieser groben Gliederung lassen sich bereits einige Anforderungen an Server und Client ableiten. So muss der Server eine dynamische Szene laden und repräsentieren sowie die der Szene zugrunde liegende Simulation berechnen können. Weiterhin muss er imstande sein, die an einen Client zu übertragenden Informationen zu selektieren, kodieren und komprimieren. Der Server entscheidet, welche und wieviele Daten ein Client erhält, wofür er die Leistungsmerkmale der Clients empfangen und verwalten muss. Somit ist eine Option zur Kommunikation zwischen Server und Client erforderlich. Aufgrund der Adaption der Daten muss der Server ein progressives Format bereitstellen und selbiges infolge von Benutzerinteraktionen eventuell zur Laufzeit für modifizierte Elemente der Szene erzeugen. Dem Client fällt die Aufgabe zu, die eingehenden Daten zu dekodieren und dekomprimieren sowie vollständige Informationen in die Szene einzufügen. Weiterhin muss er in der Lage sein, einfache Animationen auf Kommando des Servers selbstständig zu berechnen und letztendlich die Szene zu visualisieren.

Anhand dieser Anforderungen kann nun ein aufgabenorientieres Architekturmodell von Client und Server erstellt werden, wobei jeder Komponente des Modells ein Teil der Aufgaben zufällt. Einige der Komponenten existieren auf Client und Server, andere dagegen sind client- beziehungsweise serverspezifisch. Obgleich eine Komponente vielleicht von beiden Instanzen benötigt wird, ergeben sich doch jeweils unterschiedliche Anforderungen. Dies ist durch die Kommunikation mit anderen Komponenten bedingt.

Zu den auf Client und Server vorhanden Komponenten gehören die Szenen-, die Plugin-Manager-, die Codec-Manager-, die Netzwerk- und die Datei-System-Komponente. Die Szenen-Komponente bildet das Herz der Architektur von Server und Client und repräsentiert eine große, dynamische und interaktive Szene. Die Plugin-Manager-Komponente dient der Verwaltung und Auswahl sogenannter I/O Plugins. Jedes Plugin unterstützt genau ein Dateiformat, welches er lesen und schreiben kann. Während der Server die Plugin-Manager-Komponente dazu verwendet, eine Szene zu laden oder zu speichern, erlaubt die betreffende Komponente dem Benutzer auf Seite des Clients, bei Bedarf eine lokale Kopie der Szene anzulegen und später zu bearbeiten. Ähnlich der Plugin-Manager-Komponente verwaltet und selektiert die Codec-Manager-Komponente einzelne Codecs. Ein Codec ist spezifisch auf die Kodierung und Dekodierung eines bestimmten Datentyps ausgelegt. Der Server macht von der Codec-Manager-Komponente für die Kompression der zu übertragenden Informationen Gebrauch. Der Client dagegen verwendet besagten Baustein zur Dekompression der eingehenden Daten. Die Netzwerk-Komponente ermöglicht den Datenaustausch zwischen Client und Server. Die Datei-System-Komponente kapselt das zugrunde liegende Dateisystem. Dies ist einerseits wegen der Synchronisation simultaner Dateizugriffe erforderlich und andererseits, weil einige Endgeräte wie PDAs nicht über eine Festplatte verfügen.

Die serverspezifischen Komponenten entsprechen der Scheduler-, der Multiplexer-, der Client-Datenbank-, der Simulation- und der Progressive-Generator-Komponente. Die Scheduler-Komponente selektiert die an die Clients zu übertragenden Informationen. Der Multiplexer-

Komponente obliegt es dann, die Daten für den Transfer aufzubereiten. Sie entscheidet auch, welche und wieviele Informationen ein Client erhält. Die dazu notwendigen Angaben über die Leistungsmerkmale der Clients verwaltet die Client-Datenbank-Komponente mitsamt einigen anderen Daten. Die Simulation-Komponente berechnet die Simulation und berücksichtigt dabei eventuell eintreffende Benutzerinteraktionen. Die Progressive-Generator-Komponente konvertiert ein Element der Szene zur Laufzeit in ein progressives Format.

Bei den clientspezifischen Komponenten handelt es sich um die Demultiplexer-, die Animation- sowie die Präsentation-Komponente. Die Demultiplexer-Komponente dekodiert und synchronisiert die eingehenden Daten und fügt sie in die Szene des Clients ein. Die Animation-Komponente startet und begleitet eine durch die Simulation-Komponente initiierte Animation auf Seiten des Clients. Die Präsentation-Komponente dient der Interaktion mit dem Benutzer. Sie visualisiert die Szene und leitet mögliche Benutzereingaben weiter. Zudem hilft sie bei der Bestimmung der Area-of-Interest eines Clients, da sie bei der Navigation des Anwenders eine zentrale Rolle spielt.

Nach der Identifikation der Aufgaben der einzelnen Komponenten diskutiert Abschnitt 6.2 die Kommunikation zwischen den Bausteinen des Architekturmodells. Hierbei ist sowohl zwischen dem Austausch von Daten und Befehlen als auch zwischen lesenden und schreibenden Zugriffen zu unterscheiden. Im Falle geräteübergreifender Kommunikation wird davon ausgegangen, dass die Netzwerk-Komponente einen hinsichtlich des Netzwerks transparenten Nachrichtenaustausch ermöglicht. Abschnitt 6.2 gliedert nun die Spezifikation des Nachrichtenaustauschs in einzelne Vorgänge und beginnt mit dem Laden und Speichern einer Szene auf Server oder Client.

Auslösende Kraft ist hier der Benutzer eines Clients oder der Administrator des Servers. Mittels der Plugin-Manager-Komponente selektiert der betreffende ein für das gewünschte Format geeignetes I/O Plugin, welches die Szene entweder in die Szenen-Komponente einliest oder aus ihr herausschreibt. Zuvor mus das I/O Plugin sich aber um die Erlaubnis der Datei-System-Komponente in Form eines Dateihandles bemühen. Während des Datentranfers verwendet das I/O Plugin die Codec-Manager-Komponente, um spezifische Codecs für die Kodierung beziehungsweise Dekodierung der anfallenden Daten auszuwählen.

Sobald eine Szene geladen ist, kann der Server mit ihrer Übertragung an einen registrierten Client beginnen. Die Scheduler-Komponente selektiert hierfür mittels eines Area-of-Interest Konzepts die zu übertragenden Elemente einer Szene. Die Informationen über die Area-of-Interest eines Clients findet die Scheduler-Komponente in der Client-Datenbank-Komponente. Nach der Identifikation der Elemente analysiert die Multiplexer-Komponente die mit einem Element anfallenden Datenströme und entscheidet, welche und wieviele Daten ein Client erhält. Auch sie greift auf die Client-Datenbank-Komponente zurück, um die Leistungsmerkmale eines Clients zu erfragen. Anschließend kodiert sie die ausgewählten Informationen mittels der Codec-Manager-Komponente und sendet die Daten an die Demultiplexer-Komponente des Clients. Diese dekodiert die Informationen mit Hilfe der lokalen Codec-Manager-Komponente und trägt visualisierbare Elemente in die Szene des Clients ein.

Die Visualisierung der Daten auf dem Client obliegt der Präsentation-Komponente, welche die Daten aus der Szenen-Komponente liest und sie dem Benutzer anzeigt. Nun beginnt ein

interaktiver Prozess, da der Benutzer eventuell durch die Szene navigieren und sie beeinflussen möchte. Jede Manipulation der Szene teilt die Präsentation-Komponente des Verursachers zunächst der Simulation-Komponente des Servers mit. Selbige löst die Interaktionen auf und sendet während der Berechnung der Simulation Kommandos an die Animation-Komponenten der Clients, damit diese die Veränderungen in Form definierter Animationen nachvollziehen.

Nach dem Laden und Übertragen der Szene ist die Visualisierung der Szene auf dem Client der nächste Schwerpunkt. Wie in Abbildung 6.4 zu sehen, stellt dabei die Präsentation-Komponente einen entscheidenden Faktor dar: Sie liest die Daten aus der Szenen-Komponente und zeigt diese auf dem Endgerät des Benutzers an. Weiterhin bietet sie im Rahmen einer geräteübergreifenden Kommunikation mit der Simulation-Komponente die Möglichkeit zur Beeinflussung der Simulation, welche die dynamischen Vorgänge der Szene berechnet. Die Entscheidung zwischen einer Daten- beziehungsweise Kommandoverbindung ist vielleicht an dieser Stelle am schwierigsten, da die Präsentation-Komponente einerseits lediglich eine Matrix mit der Aufforderung zur Transformation eines Elements, andererseits aber auch komplexere simulationsspezifische Informationen übermitteln kann. In Abbildung 6.4 sind deshalb beide Kanäle verzeichnet. Zusätzlich ist ebenfalls ein lesender Zugriff von Seiten der Präsentation-Komponente erlaubt, um beispielsweise den aktuellen Status der Simulation zu erfragen. Weil neben der Interaktion mit dem Benutzer eine weitere Aufgabe der Präsentation-Komponente in der Fütterung der Client-Datenbank-Komponente mit Daten besteht, ist zwischen den Komponenten eine geräteübergreifende Datenverbindung eingetragen.

Während der Simulation liest die Simulation-Komponente den aktuellen Status der Szenen-Komponente und berechnet darauf basierend den nächsten Zeitschritt der Simulation. Eventuelle Änderungen registriert sie wieder in der Szenen-Komponente des Servers. Weiterhin wertet sie für die betroffenen Elemente einfache Animationsbefehle aus und sendet diese in einer ebenfalls geräteübergreifenden Kommandoverbindung an die Animation-Komponente des Clients. Die Animation-Komponente verwendet die in der Szenen-Komponente eingetragenen Animationsvorschriften, um die Elememte der Szene zu transformieren, d.h. es wird sowohl eine lesende als auch eine schreibende Datenverbindung von Seiten der Animation-Komponente zur Szenen-Komponente benötigt.

Während die beiden ersten Abschnitte die Aufgaben der Komponenten als auch ihre Kommunikation untereinander beschreiben, stellen die folgenden Abschnitte nun die Komponenten selbst vor.

Die Aufgabe der Szenen-Komponente ist die Repräsentation einer großen und dynamischen 3D Szene, wozu sie nicht nur visuelle und verhaltensspezifische Informationen, sondern auch die räumliche Sortierung der Elemente kapselt. Kern der Repräsentation ist der in Abschnitt 5.1.1 eingeführte Animationsagent, welcher ähnlich dem Prinzip der Animationselemente das visuelle Erscheinungsbild eines Elements sowie dessen Verhalten identifiziert. Zusätzlich beinhaltet er eine Transformationsmatrix, eine achsenparallele Bounding Box (AABB) seiner Geometrie sowie je nach Lokalität client- beziehungsweise serverspezifische Informationen. Animationsagenten können nach dem bekannten Prinzip der Animationshierarchien zu einem Baum strukturiert werden. Die Agenten erben dabei durch Aufmultiplizie-

ren der Matrizen die Transformationen ihrer Vorgänger in der Hierarchie.

Die Animationsagenten verweisen für ihre visuelle Beschreibung auf einen sogenannten Elementgraphen. Das Prinzip des Elementgraphen ist identisch zu kommerziellen Szenegraphen wie etwa Open Inventor. Jeder Knoten des Elementgraphen beschreibt eine spezifische Information eines Modells, beispielsweise seine Scheitelpunkte, Normalen oder Farben. Die Knoten beinhalten die Daten aber nicht direkt, sondern adressieren stattdessen Einträge in sogenannten Pools. Ein Pool gleicht einem Sammelbecken szenenspezifischer Informationen, wobei für verschiedene Datentypen wie Farben, Texturen und Geometrien auch unterschiedliche Pools existieren. Ein gesonderter Pool beinhaltet die Elementgraphen der Szene. Pools verhindern die redundante Speicherung der Informationen einer Szene, da mehrere Animationsagenten den gleichen Elementgraphen und selbige wiederum äquivalente Pooleinträge referenzieren können.

Das Verhalten eines Animationsagenten ist über die Verknüpfung von Ereignisquellen und Ereignissenken beschrieben. Ereignisquellen entsprechen zumeist Sensoren, die eine bestimmte Konstellation der Szene identifizieren und andere Instanzen darüber benachrichtigen. Sensoren können zu einer Hierarchie kombiniert werden, mit deren Hilfe das Erkennen sehr komplexer Siutationen möglich ist. Ereignissenken bedeuten in der Regel die Aktionen, die ein Agent aufgrund eines Ereignisses ausführen soll. Das Verhalten eines Animationsagenten ist dadurch als ein Graph von Ereignisquellen und Ereignissenken definiert. Derartige Graphen existieren nur auf Seite des Servers und sind in der Simulation-Komponente gekapselt. Für die Szenen-Komponente ist lediglich eine Unterart der Aktionen von Interesse, nämlich die Animationen.

Grundlegender Baustein einer Animation sind Basisanimationen wie etwa Translation, Skalierung, Rotation oder Keyframes. Basisanimationen können zu Animationslisten kombiniert werden, die sowohl die sequentielle als auch parallele Abarbeitung ihres Inhalts erlauben. Animationslisten wiederum ermöglichen ihre Verschachtelung zu komplexen Animationen und somit die Definition aufgabenorientierter Animationen. Ähnlich den visuellen Informationen besteht auch die Gefahr redundanter Animationen, weshalb sie ebenfalls über einen speziellen Pool verwaltet werden. Die Animationsagenten verweisen daher für ihre verhaltensspezifische Beschreibung auf Einträge des betreffenden Pools. Selbiger beinhaltet genau diejenigen Animationsvorschriften, welche an die Animation-Komponente des Clients übertragen und dort von der Animation-Komponente auf Kommando der Simulation-Komponente umgesetzt werden.

Ein wichtiger Punkt vieler Applikationen wie Occlusion Culling Verfahren oder Kollisionserkennungen ist die effiziente räumliche Sortierung der Elemente einer Szene. Abschnitt 6.3.4 beschreibt hierzu ein neuartiges Konzept eines dynamischen Raumunterteilungsbaumes, auf den drei Operationen definiert sind: Das Einfügen eines Animationsagenten, das Entfernen eines Animationsagenten sowie die Transformationen eines Animationsagenten. Jede dieser Operationen kann eine Verletzung der Kapazität einer Zelle des Raumunterteilungsbaumes verursachen, weshalb eine Reorganisation des Baumes notwendig wird. Im Rahmen dieser Reorganisation sind zwei Basisoperationen definiert. Während die Divide Operation eine Zelle beim Überlauf der Kapazität δ in mehrere Tochterzellen unterteilt und selbigen die Animationsagenten zuordnet, fasst die Unite Operation einen gesamten Teilbaum wieder zu einem

einzelnen Knoten zusammen, sofern die Anzahl der Animationsagenten in dem Teilbaum die Kapazität δ einer einzelnen Zelle unterschreitet. Leider gibt es nicht den optimalen Wert für die Kapazität einer Zelle. Vielmehr muss dieser von Szene zu Szene unterschiedlich bestimmt werden. Um die Anzahl der mit der Reorganisation anfallenden Basisoperationen zu reduzieren, definiert Abschnitt 6.3.4 einen Toleranzbereich der Kapazität einer Zelle von $delta_{min}$ bis $delta_{max}$. Hierdurch wird insbesondere an Unite Operationen gespart, da diese nun erst mit dem Unterschreiten der minimalen Kapazität anstehen. Um die mehrfache Verletzung der Kapazitäten durch die Bewegung eines einzelnen komplexen Elements zu vermeiden, werden die Agenten einer Animationshierarchie in einem Hierarchieknoten gekapselt und im Sinne des Raumunterteilungsbaumes als eine Einheit verstanden. Wohingegen einzelne Animationsagenten anhand ihrer AABB in den Baum einsortiert werden, pflegt ein Hierarchieknoten eine AABB für alle seine Animationsagenten. Die resultierende AABB ist dann das Kriterium für die Zuordnung im Raumunterteilungsbaum. Die Idee der Hierarchieknoten basiert auf der Beobachtung, dass eine Animationshierarchie üblicherweise räumlich benachbarte Agenten beinhaltet, welche sich tendenziell in eine äquivalente Richtung bewegen. Weiterhin ist die Einsortierung der Animationsagenten anhand einer Temporary Bounding Box [SG96] möglich. Diese definiert hinsichtlich eines Agenten einen Raum und ein Zeitintervall, währenddessen der Agent den Raum nicht verlässt. Innerhalb des Intervalls impliziert ein Animationsagent keine Verletzung der Kapazitäten einer Zelle. Das Konzept der sich gegenseitig kompensierenden Bewegungen sammelt in der Zeitspanne zwischen zwei Simulationszeitschritten alle dynamischen Vorgänge und betrachtet bei der nunmehr lediglich pro Zeitschritt anfallenden Reorganisation ausschließlich den Nettowert der Bewegungen. Zudem bietet das Konzept die Möglichkeit paralleler Lese- und Schreibzugriffe auf die Zellknoten des Raumunterteilungsbaumes. Das Problem der Schnittelemente ist mit Hilfe des Ansatzes von Chang et al [CLC03] gelöst.

Das in Abschnitt 6.3.5 erläuterte Konzept der Renderer erlaubt die Kapselung plattformspezifischer Merkmale sowie die Realisierung beliebiger Level-of-Abstraction. Renderer stellen Komponenten dar, welche die Datenstrukturen der Szenen-Komponenten traversieren und dabei eine bestimmte Ausgabe erzeugen. Entsprechend den Strukturen gibt es Poolrenderer, Elementgraphrenderer sowie Raumrenderer, wobei erstere die Pooleinträge und letztere den Raumunterteilungsbaum traversieren. Elementgraphrenderer bearbeiten Elementgraphen. Durch die Konzeption der Renderer erlauben sie gemäß Abschnitt 5.1.1 ihren transparenten Austausch. Das hierfür erforderliche Framework basiert im Wesentlichen auf Methoden zur Traversierung der Datenstrukturen.

Nach der Repräsentation einer großen, dynamischen Szene ist der nächste Schritt die Übertragung der Szene an ein Endgerät. Hierfür ist sowohl eine Adaption als auch eine Selektion der zu übertragenden Informationen erforderlich. Für die Adaption der Daten beschreibt Abschnitt 6.4 ein invasives Konzept für die progressive Simplifizierung von Dreiecksnetzen, welches den Anforderungen aus Abschnitt 5.1.1 genügt. Das Konzept basiert auf dem Half-Edge Collapse Operator, wodurch eine Kante zwar entfernt aber keine neuen Scheitelpunkte berechnet werden. Die neuartige Fehlermetrik erlaubt die sehr schnelle Bestimmung des Fehlers eines Scheitelpunktes bei gleichzeitig guten visuellen Ergebnissen. Allerdings ist sie nicht Teil eines sauberen mathematischen Modells wie bei Hoppe [Hop96] oder Klein et al. [KLS96]. Insbesondere vernachlässigt das Konzept im Gegensatz zu Klein et al. das Einhal-

ten von Fehlerschranken. Mit jedem Anwenden des Half-Edge Collapse Operator entsteht ein vereinfachtes Dreiecksnetz. Um das ursprüngliche Dreiecksnetz wieder auf Seite des Clients rekonstruieren zu können, werden für jede Operation entsprechende progressive Daten angelegt. Glücklicherweise beanspruchen diese nur einen sehr geringen Speicherbedarf und erlauben zudem die extrem einfache und kostengünstige Verfeinerung auf einem Client. Die dabei verwendeten Datenstrukturen bieten eine konsistente Repräsentation des Modells, welche zum Zwecke der Visualisierung ohne jegliches Konvertieren oder Traversieren an eine Grafik-Bibliothek wie OpenGL übergeben werden kann. Zusatzattribute werden pro Scheitelpunkt als auch pro Fläche unterstützt und liegen auch nach der Simplifizierung noch als getrennte Datenströme vor. Hierdurch ist die spezifische Kodierung und Dekodierung der Datenströme durch geeignete Codecs gegeben. Scheitelpunkte wie auch Zusatzattribute beschreiben innerhalb der Datenströme eine zu den Kantenkollabierungen inverse Sequenz, wodurch eine Aufteilung der Ströme in einzelne Datenpakete möglich ist. Die Pakete können dann clientseitig einfach aneinander gereiht werden. Ähnliches gilt auch in Bezug auf die topologischen Informationen, wobei hier aber eine gesonderte Behandlung im Sinne der progressiven Daten erforderlich ist. Die Implementierung des Verfahrens als Elementgraphrenderer erlaubt die Verarbeitung beliebiger nach dem Konzept der Elementgraphen erstellter Modelle. Im Anschluss an die Simplifizierung identifiziert ein Elementgraph neben dem Basismesh auch die für dessen Verfeinerung notwendigen Daten.

Die in Abschnitt 6.5 beschriebene Scheduler-Komponente stellt in Bezug auf die Adaption der Informationen ein nicht-invasives Verfahren dar. Sie selektiert anhand eines visuellen Area-of-Interest Konzepts die zu übertragenden Animationsagenten. Dazu wird auf Seite des Servers jeder Client durch eine Area-of-Interest repräsentiert. Die mit einer Area-of-Interest verknüpften Parameter erhält die Scheduler-Komponente von der Client-Datenbank-Komponente, welche wiederum durch die Präsentation-Komponente des Clients in regelmäßigen Zeitabständen mit Informationen versorgt wird. Im Konzept der vorliegenden Arbeit besteht eine Area-of-Interest aus drei ineinander verschachtelten Zonen, welche jeweils ein achsenparalleles, quaderförmiges Gebiet beschreiben. Die innerste Zone umschließt dabei die Sichtpyramide des Betrachters und wird deshalb sichtbare Zone genannt. Animationsagenten, die in dieser Zone liegen, erhalten hinsichtlich der Übertragung die höchste Priorität. Die mittlere Zone ist lediglich im Falle von Bewegungen des Betrachters von Bedeutung und wird entsprechend als dynamische Zone bezeichnet. Die äußerste Zone entspricht der präventiven Zone, da sie Animationsagenten berücksichtigt, die zwar momentan außerhalb des Sichtbereiches liegen, aber bald in ihn hineingeraten könnten. Animationsagenten innerhalb der dynamischen Zone erhalten eine mittlere Priorität und Agenten innerhalb der präventiven Zone die niedrigste Priorität.

Die Idee des in Abschnitt 6.5.2 erläuterten Ansatzes beruht darauf, die Anzahl der Vergleiche zwischen den Zonen der Areas-of-Interest und den Animationsagenten zu minimieren. Hierzu werden nicht nur die räumlichen Kohärenzen der Animationsagenten in Form des Raumunterteilungsbaumes evaluiert, sondern auch die räumlichen Kohärenzen der Zonen selbst. Aus diesem Grund erfolgt über die Zonen der Clients die Konstruktion von drei hierarchischen Baumstrukturen, den sogenannten Clientbäumen. Jeder Clientbaum ist mit einer bestimmten Priorität assoziiert, weshalb alle sichtbaren Zonen dem sichtbaren Clientbaum, die dynamischen Zonen dem dynamischen Clientbaum und die präventiven Zonen dem prä-

ventiven Clientbaum zugeordnet werden. Nach dem Aufbau eines Clientbaumes wird die Szene beziehungsweise der Raumunterteilungsbaum gegen den Clientbaum getestet und so die Animationsagenten identifiziert, die innerhalb einer Zone liegen. Da mit jedem Baum eine Priorität und mit jeder Zone ein Client verbunden ist, ergibt sich eine genaue Aussage, welcher Agent mit welcher Dringlichkeit an einen Client übertragen werden muss. Die Auftrennung in drei verschiedene Clientbäume resultiert in kleineren, wesentlich einfacheren Bäumen und damit in einer schnelleren Auswertung. Außerdem müssen beim Testen der Clientbäume mit höherer Priorität lediglich die Bereiche der Szene berücksichtigt werden, die während der vorangehenden Analyse des Clientbaumes mit der nächst niedrigeren Dringlichkeitsstufe mit einer entsprechenden Priorität versehen wurden.

Wie in Abschnitt 6.5.5 erläutert, ermöglichen die von der Scheduler-Komponente ermittelten Prioritäten der Animationsagenten nicht nur deren Einstufung bezüglich der Übertragung, sondern sie bilden außerdem die Grundlage für ein effizientes Out-of-Core Konzept. Ein derartiges Konzept ist für sehr große Szenen von Bedeutung, die selbst die Speicherkapazität des Servers übersteigen. Das Problem eines Out-of-Core Konzepts besteht in der Identifikation derjenigen Informationen, die im Falle eines Speicherüberlaufes zu verwerfen sind. Steht dagegen genügend Speicher zur Verfügung, so steht die schwierige Entscheidung an, welche Informationen denn nun einzulagern sind. Da fast alle zeitkritischen Operationen des Servers auf den Animationsagenten basieren, die sich in mindestens einer sichtbaren Zone befinden, entsprechen die zuvor bestimmten Prioritäten auch der Dringlichkeit bezüglich der Aus- beziehungsweise Einlagerung der Animationsagenten.

Nach der Selektion eines Animationsagenten durch die Scheduler-Komponente identifiziert die in Abschnitt 6.6 beschriebene Multiplexer-Komponente die mit dem Agenten assoziierten Datenströme. Dabei handelt es sich um den Animationsagenten selbst, seinem Elementgraphen sowie den vom Elementgraphen adressierten Pooleinträgen. Da sich mehrere Animationsagenten den gleichen Elementgraphen teilen und wiederum verschiedene Elementgraphen äquivalente Pooleinträge indizieren können, besteht die Gefahr, dass ein Elementgraph beziehungsweise ein Pooleintrag in Bezug auf die Übertragung an einen Client mehrere Prioritäten aufweist. In diesem Fall ist die höchste Priorität maßgebend. Die Multiplexer-Komponente erstellt für jeden Client einen Abhängigkeitsgraphen, dessen Knoten durch die Agenten, ihren Elementgraphen sowie deren Pooleinträgen gebildet werden. Sie fügt allerdings nur diejenigen Pooleinträge in den Abhängigkeitsgraphen ein, die auch von dem Client unterstützt werden. Die Multiplexer-Komponente evaluiert den Abhängigkeitsgraphen, um die Informationen in einer optimalen Reihenfolge an den Client zu übertragen. Während die Animationsagenten und Pooleinträge sehr kleine, kompakte Informationen darstellen, erfordern insbesondere die Pooleinträge eine Kodierung ihrer Daten. Die Multiplexer-Komponente selektiert hierzu mittels der Codec-Manager-Komponente einen für den Pooleintrag geeigneten Codec, dem sie die Daten zur Komprimierung übergibt. Dabei fällt der Multiplexer-Komponente die Entscheidung zu, welche Datenmengen ein Client erhält. Sie verwendet hierfür einen normierten Benchmarkindex, welcher die Leistungsmerkmale des Clients repräsentiert und als prozentualer Faktor auf die Anzahl der Daten angesetzt wird.

Auf Seite des Clients empfängt die Demultiplexer-Komponente die eingehenden Informationen und erstellt ähnlich der Multiplexer-Komponente ebenfalls einen Abhängigkeitsgraphen. Mit dessen Hilfe synchronisiert sie die einzelnen Datenströme und ist in der Lage, die Daten-

ströme einem Animationsagenten zuzuordnen und so seine Vollständigkeit zu überprüfen. Sobald ein Animationsagent als visualisierbar identifiziert wird, trägt die Demultiplexer-Komponente den Agenten in die Szenen des Clients ein.

Für die Visualisierung der Szene zeichnet sich die in Abschnitt 6.7 erläuterte Präsentation-Komponente verantwortlich. Obwohl ein Client üblicherweise nur eine Teilmenge der Szene verwaltet, ist dennoch eine ausgabesensitive Visualisierung der Informationen erforderlich. Abschnitt 6.7.2 beschreibt dazu ein neues, hardwarebasiertes Occlusion Culling Verfahren, welches im Gegensatz zu Bartz et al. [BMH99] keine Features verwendet, die nicht zum Standard derzeitiger Grafikbibliotheken gehören. Das Verfahren beruht auf der effizienten Auswahl, vollständig dynamischen Auswahl von Occludern, die ähnlich zu Bittner et al. [BHS98] in einem Occlusion Tree vereinigt werden. Anschließend erfolgt ein hierarchischer Test der Szene gegen den Occlusion Tree. Der Vorteil des präsentierten Verfahrens liegt neben seiner Effizienz in der Parametrisierung seiner Präzision und damit auch seiner Laufzeit, wodurch eine Adaption an die Rechenleistung eines Endgerätes ermöglicht wird.

Neben der Ausgabe der Szene obliegt der Präsentation-Komponente ebenfalls die Verarbeitung der Benutzereingaben. Unter anderem hat sie die Option, anhand der Eingaben den Navigationspfad des Benutzers nachzuvollziehen. Aus diesem Grund ist die Präsentation-Komponente Teil des bereits zuvor erwähnten Area-of-Interest Konzepts. Sie verwendet die analysierten Standorte des Betrachters allerdings nicht nur im Sinne einer Historie, sondern versucht, daraus den zukünftigen Weg des Benutzers vorherzubestimmen. Die Ergebnisse entsprechen den Dimensionen der Zonen einer Area-of-Interest, d.h. Größe und Lage der einzelnen Zonen einer Area-of-Interest hängen von der Navigationsgeschwindigkeit und Navigationsrichtung des Anwenders ab. Das Konzept orientiert sich dabei an der Erkenntnis von Teler et al. [TL01], wonach eine Navigation sich in wenige grundlegende Situationen diskretisieren lässt. Der in Abschnitt 6.7.1 beschriebene Feldtest bestätigt im Wesentlichen die Angaben von Teler et al. und gliedert eine Navigation in positionsbezogene Handlungen, in geradlinige Bewegungen sowie in bogen- beziehungsweise kurvenförmige Bewegungen. Für jede dieser Navigationsformen wird eine spezifische Vorgehensweise für die Prekalkulation des Betrachterpfades vorgestellt. Die resultierenden Dimensionen der Zonen übermittelt die Präsentation-Komponente der Client-Datenbank-Komponente des Servers.

Für die komfortable Navigation des Benutzers sorgt die in Abschnitt 6.7.3 erklärte Navigationsschnittstelle, deren Konzept auf bereits verfügbaren 3D Spielen fußt. Demnach wird die Bildfläche des Endgerätes in verschiedene Felder unterteilt, wobei jedes Feld einer bestimmten Form der Navigation entspricht. Das Selektieren eines Feldes per Maus oder PDA Stift resultiert in einer entsprechenden Bewegung, wobei die Position innerhalb des Feldes die Geschwindigkeit der Bewegung beeinflusst. Die Idee hinter dem Konzept beruht darauf, eine allgemeine Schnittstelle für sämtliche Endgeräte anzubieten. Auf diese Weise kann der Benutzer die auf einem Gerät erworbenen Erkenntnisse auf ein anderes Gerät übernehmen.

Die in Abschnitt 6.8 diskutierte Simulation-Komponente berechnet die der Szene zugrunde liegende Simulation. Analog zur Intention von Funkhouser [Fun95] verwendet sie die durch die Scheduler-Komponente ermittelten Prioritäten der Animationsagenten, um den mit der Synchronisation einer dynamischen Szene anfallenden Kommunikationsaufwand zu reduzieren. Sie sendet einem Client nur die Aktualisierungen derjenigen Animationsagenten zu, die

sich innerhalb der Area-of-Interest des Clients befinden.

Kapitel 7

Implementierung

In diesem Kapitel wird die Umsetzung des Konzepts aus Kapitel 6 beschrieben. Aufgrund des objektorientierten beziehungsweise komponentenbasierten Ansatzes finden sich in den einzelnen Abschnitten Hinweise auf verwendete Software-Patterns. Die Realisierung der Simulation- und Animation-Komponente wird nicht beschrieben, da beide je nach Anwendung unterschiedlich ausfallen können. Die Komponenten wurden prototypisch für ein Online Schachspiel implementiert, worauf das folgende Kapitel eingeht.

7.1 Verwendete Werkzeuge

Dieser Abschnitt beschäftigt sich mit den während der Implementierung verwendeten Werkzeugen. Insgesamt erfolgte die Realisierung mit der Programmiersprache C++.

7.1.1 STL

Die *Standard Template Library* (STL) ist eine C++ Bibliothek, die Implementierungen für grundlegende Algorithmen und Datenstrukturen wie etwa verkettete Listen, Mengen, Key-Value Maps oder Vectors enthält. Beinahe sämtliche Implementierungen beruhen auf dem Template Pattern. Es gibt unterschiedliche Implementierungen der STL, unter anderem von Microsoft und SGI. Innerhalb der Arbeit wurde die SGI Implementierung bevorzugt, da in der Microsoft Realisierung einige Effekte auftreten, die nicht der Spezifikation entsprechen.

7.1.2 Ace

Das *Adaptive Coomunication Environment* (ACE) [Sch] ist ein frei verfügbares, objektorientiertes Framework für nebenläufige Kommunikationssoftware. Innerhalb der vorliegenden

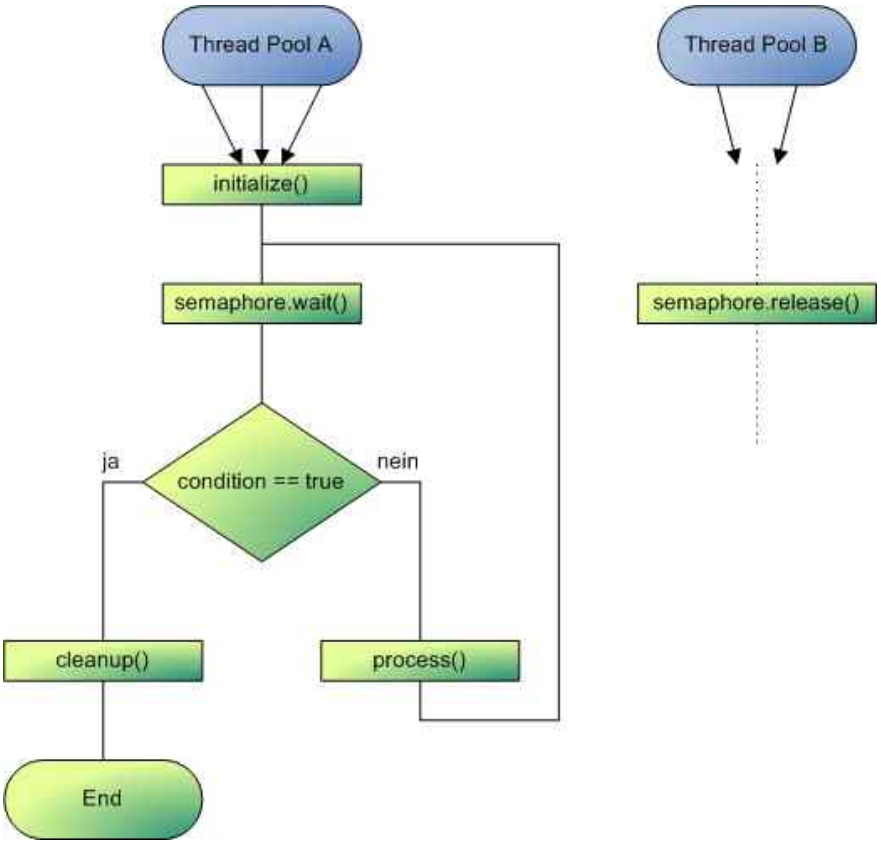

Abbildung 7.1: *Eine Möglichkeit, plattformspezifische Threadtechniken zu vermeiden, beruht auf dem Einsatz von Semaphoren.*

Arbeit kommt es insbesondere im Rahmen der Netzwerkkommunikation und des Multithreadings sowie für den Schutz kritischer Programmbereiche in Form von Semaphoren und Mutices zum Einsatz. ACE ist auf alle wichtigen Plattformen erhältlich und bietet unter anderem einen Windows CE Support, d.h. ACE kann auch auf Pocket PCs verwendet werden.

Auch wenn ACE mit Hilfe des Frameworks eine Abstraktion der Schnittstelle zu der zugrunde liegenden Plattform erreichen möchte, ist selbiges nur unter Einschränkungen möglich. Beispielsweise erlaubt Windows keine Signaltechniken der Threads, weshalb ein Thread nicht ohne weiteres unterbrochen und wieder fortgesetzt werden kann. An dieser Stelle ist es also erforderlich, entsprechende Techniken zu vermeiden und alternative Lösungen zu verwenden. Abbildung 7.1 illustriert ein derartiges Pattern, welches innerhalb der Implementierung häufig verwendet wird.

Das von Abbildung 7.1 behandelte Problem liegt darin, dass wichtige Betriebssysteme wie etwa Windows in Bezug auf Threads keine *stop*, *resume* oder *kill* Methoden anbieten. Nichtsdestotrotz soll ein aktives Polling vermieden werden, d.h. ein Thread sollte nicht in regelmäßigen Abständen nach eventuellen Aufträgen Ausschau halten, sondern nur dann Rechenkapazität beanspruchen, wenn auch wirklich Arbeit anliegt. Aufgrund möglicher Parallelität wird gemäß dem *Thread Pool Pattern* eine Menge von Threads aus einem Thread Pool auf ein Problem angesetzt. Diese Threads durchlaufen wie in Abbildung 7.1 dargestellt erst einmal eine Initialisierungsphase und blockieren dann an einem *Semaphor*. Ein Semaphor bietet

gegenüber einem *Mutex* den Vorteil, von einem beliebigen Thread wieder freigegeben werden zu können. Außerdem besteht über den Zähler des Semaphors die Möglichkeit, mehrere Threads zu deblockieren. Weiterhin kann diese Freigabe zu einem beliebigen Zeitpunkt geschehen und nicht nur dann, wenn ein Thread gerade am Semaphor wartet. Jedesmal wenn also ein Auftrag ansteht, erhöhen die Threads aus dem Pool B den Zähler N des Semaphors um eins. Ist $N > 0$, dann können N Threads den Semaphore passieren und somit die N Aufträge abarbeiten. Eventuelle kritische Bereiche innerhalb der Aufträge müssen durch weitere Mutices geschützt werden. Beim Beenden des Systems wird der Semaphor vollständig freigegeben, so dass alle Threads passieren können. Dort treffen sie auf eine Bedingung, die sie zum Verlassen der Schleife auffordert. Außerhalb der Schleife können die Threads dann sauber beendet und anstehende Aufräumaktionen durchgeführt werden.

7.1.3 QT

Qt [Tro] ist ein Produkt der Firma Trolltech und stellt ein Framework für die Gestaltung und Implementierung von Benutzeroberflächen dar. Sein Ursprung liegt im Jahr 1995. Ähnlich wie bei allen übrigen Werkzeugen ist ein entscheidendes Merkmal von Qt seine Verfügbarkeit auf den wichtigsten Plattformen. Zudem bietet es die Möglichkeit, mit Hilfe einer speziellen Fensterklasse einen OpenGL Kontext zu erzeugen und zu verwenden. Diese Eigenschaft macht sich die Präsentation-Komponente zunutze, welche die Kommunikation mit dem Benutzer kapselt. Grundlegender Baustein von Qt ist die Klasse *QWidget*, welche die Basisfunktionen aller Fenster bereitstellt. Jedes Fenster entspricht einem Container, in den eine Hierarchie weiterer Oberflächenelemente wie etwa Knöpfe oder Menüs eingefügt werden können. Die Lage und Position dieser Elemente wird über verschiedene Layouts kontrolliert.

Ein Problem von Qt liegt in seinem Konzept der Ereignisverwaltung. Leider verwendet Qt eine globale Ereignisliste, die nur von einem einzelnen Thread bearbeitet werden kann. Dies ist insofern problematisch, weil sich der gleiche Thread auch für die Erzeugung sowie die Aktualisierung der Oberfläche und damit für das Rendern der Szene verantwortlich zeichnet. Erschwerend kommt hinzu, dass nur derjenige Thread einen OpenGL Kontext verwenden darf, der selbigen erzeugt hat. D.h. es gibt in Qt einen Thread, welcher das OpenGL Fenster und dessen OpenGL Kontext erzeugt, die Benutzeroberfläche verwaltet und zudem die Szene rendert. Als Konsequenz können niedrige Frameraten bei der Ausgabe einer Szene zu Blockaden der Oberfläche führen. Auch der Einsatz der Klasse *QTimer* zur getriggerten Visualisierung in regelmäßigen Zeitabständen bietet nicht die eigentlich gewünschte Multithread Lösung, da QTimer nach einem Zeitscheibenverfahren Rechenzeit von dem Ausgangsthread abzweigt.

7.1.4 OpenGL

OpenGL [SGIc] ist ein *Application Programming Interface* (API) für die Entwicklung von interaktiven 2D und 3D Grafikapplikationen. Es wurde 1992 eingeführt und entspricht mittlerweile einem industriellen Standard. Zwei große Vorteile von OpenGL liegen in dessen Verfügbarkeit auf nahezu allen wichtigen Plattformen sowie in der Unterstützung durch die

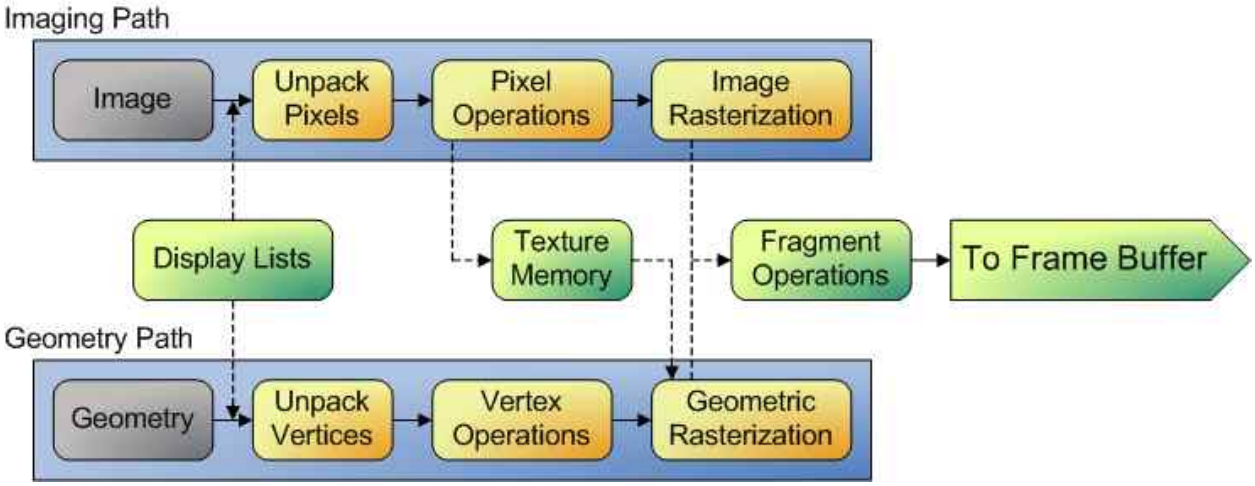

Abbildung 7.2: *Die Ausgabepipeline von OpenGL kombiniert geometrische und bildbasierte Informationen.*

wichtigen Grafikchip Hersteller. Außerdem ist eine sehr gute Dokumentation der Schnittstelle vorhanden. Ein Problem von OpenGL stellt die doch ein wenig schleppende Standardisierung neuer Technologien dar. So stehen viele sinnvolle Features wie etwa die NVIDIA-Occlusion-Query nur als Erweiterungen zur Verfügung, weshalb die Gefahr der Abhängigkeit zu einer bestimmten Plattform oder Grafikhardware besteht.

Das Konzept von OpenGL beruht auf der in Abbildung 7.2 dargestellten Ausgabepipeline, welche sowohl geometrische als auch bildbasierte Informationen in den *Frame Buffer* rasterisiert. Dabei arbeitet OpenGL zustandsorientiert, d.h. ein einmal gesetzter Zustand bleibt solange gültig, bis er durch eine neue Information des gleichen Typs überschrieben wird. Obgleich OpenGL auf vielen Plattformen zur Verfügung steht, ist seine API innerhalb der vorliegenden Arbeit nochmals durch die in Abschnitt 6.3.5 beschriebenen Renderer gekapselt. Beispielsweise gibt es einen auf OpenGL basierenden Elementgraphrenderer, der für die Ausgabe des visuellen Erscheinungsbildes eines Animationsagenten sorgt. Prinzipiell ist aber ebenso die Implementierung eines DirectX Renderers denkbar.

7.1.5 Glut und Freeglut

Das *OpenGL Utility Toolkit* (GLUT) [Inc] ist ähnlich wie Qt eine Oberflächenbibliothek, die allerdings speziell auf die Implementierung von OpenGL Applikationen ausgelegt ist. GLUT wurde von Mark Kilgard entwickelt und ist für eine Vielzahl unterschiedlicher Plattformen erhältlich. Neben dieser Verfügbarkeit gibt es in der vorliegenden Arbeit zwei weitere Gründe, um die GLUT Bibliothek einzusetzen. Zum einen bietet GLUT eine Lösung für das in Abschnitt 7.1.3 beschriebene Problem der Ereignis- und Fensterverwaltung. Zum anderen stellt es gegenüber Qt eine in Bezug auf die Rechenzeit kostengünstigere Alternative dar. GLUT kommt innerhalb der vorliegenden Arbeit für die Implementierung des Servers zum Einsatz. Da die primäre Aufgabe des Servers in der Verteilung und Verwaltung der 3D Szenen besteht, ist hier eine komfortable Oberflächenbibliothek wie etwa Qt nicht erforderlich. Eventuelle Visualisierungen auf Seite des Servers dienen eher der Kontrolle durch den

Administrator. Aufgrund der hohen Belastung der Servers bedeutet eine aufwändige Benutzeroberfläche eher eine Blockade denn eine wirklich Hilfe. Im Wesentlichen ist der Server als Konsolenapplikation ausgelegt, die als Hilfestellung das Einblenden eines GLUT Fensters erlaubt.

Viele Oberflächenbibliotheken wie auch GLUT verbinden mit einer Applikation ein Fenster. Wird dieses Fenster geschlossen, so gilt die Applikation als beendet und der Prozesse fährt herunter. Für eine zwischenzeitliche Kontrollansicht auf Seiten des Servers ist das leider nicht geeignet. Eine Lösung bietet das frei verfügbare *freeglut* Projekt [FRE]. Dabei handelt es sich um eine Weiterführung der GLUT Bibliothek. Ein wesentlicher Unterschied besteht darauf, dass die *glutMainLoop* Methode der freeglut Implementierung jederzeit beendet und wieder neu gestartet werden kann, ohne ein Beenden des Prozesses mit sich zu bringen.

7.1.6 DjVu

DjVu [HCB+99] ist eine progressive Kompressionstechnik für Dokumente mit Texten und Bildern. DjVu wird seit 1996 von AT&T Labs entwickelt und von LizardTech Inc. vertrieben. Es bietet in Bezug auf Farbbilder eine gegenüber JPEG und GIF fünf- bis zehnfach höhere Kompressionsrate sowie hinsichtlich monochromen Bildern eine drei- bis achtfach höhere Kompressionsrate als TIFF (siehe Abbildung 7.3).

Ein großer Vorteil von DjVu ist die freie Verfügbarkeit des Source Codes. Darüber was es in der vorliegenden Arbeit möglich, den im Vergleich zum Gesamtpaket eher kleinen Teil zu entnehmen, der sich mit der progressiven Kompression einzelner Bilder und nicht ganzer Dokumente beschäftigt. Der entsprechende Programmcode wurde in eine Bibliothek für die Umwandlung von TGA, PPM und PGM Bildern in das DjVu-spezifische IW44 Bildformat eingefügt. Das IW44 Format wird innerhalb der Arbeit für die Repräsentation der Texturen verwendet. Durch die Erweiterung des Quellcodes mit MMX Befehlen [1] bietet diese waveletbasierte Bibliothek neben der hohen Kompression auch sehr gute Laufzeiteigenschaften. Ein weiteres positives Merkmal ist die Option, sowohl die Anzahl als auch die Größe der einzelnen progressiven Pakete beziehungsweise Chunks eines Bildes vorgeben zu können. Hierdurch kann der Aufwand zur Übertragung eines Bildes an die aktuell vorhandene Bandbreite angepasst werden.

Mit dem ebenfalls waveletbasierten JPEG2000 existiert zwar ein weiteres progressives Bildformat, allerdings besteht hier das Problem der freien Verfügbarkeit. Obgleich es durchaus einige offene Bibliotheken gibt, beschränken diese sich ausschließlich auf die Kodierung eines Bildes. Die progressive Dekodierung ist dagegen nur gegen Lizenzen erhältlich.

[1] MMX steht für *Multimedia Extensions* und ist eine in Intel- und andere x86-kompatible Prozessoren eingebaute Hardwareerweiterung. Die Idee beruht darauf, einfache mathematische Befehle durch hohe Registerbreiten zu parallelisieren. Ideal ist MMX vor allem für Bildverarbeitungen wie etwa Wavelets.

Abbildung 7.3: *Ein Vergleich des IW44 Bildformats von DjVu mit JPEG. Das Original rechts unten entspricht einem JPEG Bild mit ca. 100000 Bytes. Dagegen liefert bereits das DjVu Bild links oben mit ca. 4000 Bytes eine sehr gute Annäherung. Die folgenden Bilder zeigen den Hauseingang jeweils nach dem Hinzufügen eines Chunks von ca. 4000 Bytes.*

7.2 Datenverwaltung

In diesem Abschnitt werden grundlegende Aspekte der Datenverwaltung erörtert. Unter anderem geht es um Problemstellungen wie etwa Daten in Dateien ausgelagert und referenziert sowie Speicherblöcke effizient verwaltet werden können.

7.2.1 Implementierung der Speicherverwaltung

Aufgrund des begrenzten Speicherbereiches der Systeme ist es im Falle großer Szenen immer wieder erforderlich, einen Teil der Daten zu verwerfen bevor neue Informationen eingefügt werden können. Oftmals handelt es sich dabei um kleine Datenobjekte wie etwa den Knoten des Szenegraphen. Hier entsteht der hohe Datenaufwand nicht durch die Größe der Objekte, sondern durch deren Quantität. Ein Problem der Speicherverwaltung ist die zunehmende Fragmentierung des Speichers. Aus diesem Grund starten die Betriebssysteme ab einer bestimmten Fragmentierung eine Aufräumaktion, welche sich beispielsweise im Falle von Windows beim Löschen vieler Objekte mehrere Minuten hinzieht. Anhand des Windows Task Managers kann dann verfolgt werden, wie das Betriebssystem den Speicher kilobyteweise freigibt. In einem derartigen Fall kann von einer Echtzeitanforderung nicht mehr die Rede sein.

Die Lösung besteht darin, einmalig einen großen Speicherblock zu allokieren und selbigen dann eigenständig zu verwalten. In der aktuellen Implementierung wird dabei aus Geschwindigkeitsgründen keine Rücksicht auf eine mögliche Fragmentierung genommen. Der Schwerpunkt des Verfahrens beruht auf der schnellen Reservierung eines für einen Datensatz adäquaten Speicherblocks beziehungsweise auf der effizienten Freigabe des belegten Speichers.

Die Vorgehensweise ist mit einer Hash Table bestehend aus 32 Einträgen vergleichbar. Jeder Eintrag repräsentiert eine bestimmte Speichergröße und beinhaltet eine verkettete Liste, in die freie Speicherblöcke mit entsprechender Größe eingefügt werden. Die Anzahl der Listen in der Hash Table ist durch die Registerbreite der derzeit gängigen 32 Bit Betriebssysteme vorgegeben. Beispielsweise können unter Windows Prozesse maximal zwei Gigabyte an Speicher beanspruchen [2]. Ein freier Speicherblock wird in diejenige Liste einsortiert, welche dem höchstwertigstem Bit seiner Größe entspricht. Beispielsweise sind nach Abbildung 7.4 die freien Blöcke mit 47 beziehungsweise 53 Bytes in die Liste Nummer fünf einzutragen, da das fünfte Bit dem höchstwertigsten Bit beider Zahlen entspricht. Die Einträge innerhalb der Listen sind nicht nach Größe sortiert. Eine Liste garantiert lediglich, dass in ihr vermerkte freie Speicherblöcke mindestens die Größe des zur Liste korrespondierenden höchsten Bits besitzen. Für jede Liste existiert ein Flag beziehungsweise ein einzelnes Bit, ob die Liste über Einträge verfügt oder nicht. Diese Flags sind zu einem 32 Bit Speicherwort zusammengefasst.

Ein Speicherblock besteht aus einem Header und dem eigentlichen Datenbereich. Der Header hat eine Länge von 16 Bytes und beinhaltet die Größe des Blocks, den vorangehenden Block im Hauptspeicher sowie den Vorgänger beziehungsweise Nachfolger innerhalb der verketteten Liste, sofern es sich um einen freien Speicherblock handelt. Der folgende Block im Hauptspeicher kann über die Größe des Blocks ermittelt werden. Aufgrund des Headers hat jeder Block mindestens eine Länge von 16 Bytes, weshalb die vier ersten Listen innerhalb der Hash Table niemals über Einträge verfügen.

Auf die Hash Table sind zwei grundlegende Operationen definiert, nämlich das Belegen und die Freigabe von Speicherblöcken. Zu Beginn ist der Hauptspeicher vollständig leer, d.h. es existiert nur ein einzelner Block der entsprechend seiner Größe in einer der Listen vermerkt

[2] Zwei Gigabyte entsprechen zwar eigentlich nur 31 Bit, aber das Verfahren kann bis zu vier Gigabyte verwalten.

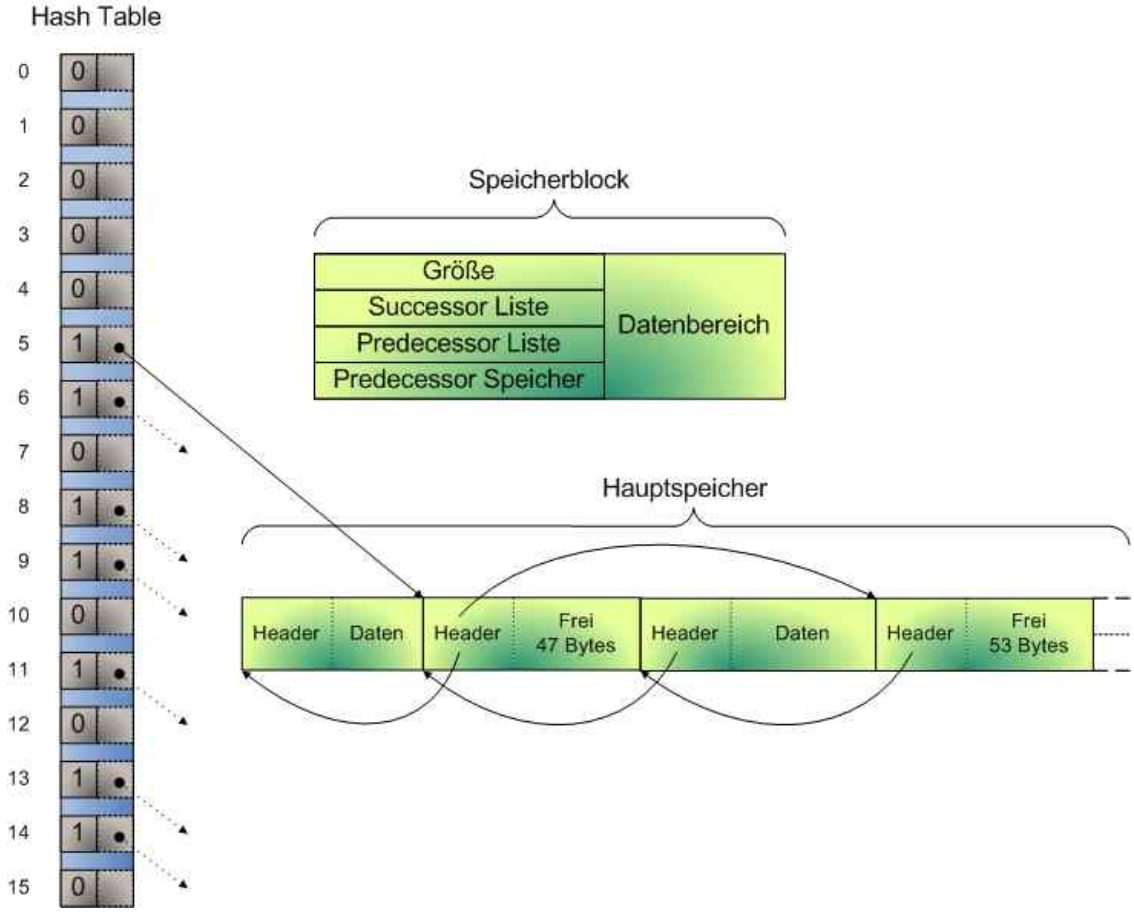

Abbildung 7.4: *Die Struktur zur Verwaltung des Hauptspeichers hier auf 16 Bit reduziert.*

ist. Bei der Reservierung eines Blockes wird die Position σ des höchsten gesetzten Bits in der geforderten Größe berechnet. Die Suche nach σ innerhalb eines 32 Bit Wertes kann durchaus zeitintensiv sein. Aus diesem Grund wird der in Abbildung 7.5 illustrierte Algorithmus verwendet. Zunächst erfolgt über zwei Bedingungen die Bestimmung desjenigen Bytes, welches das höchste gesetzte Bit enthält. Innerhalb dieses Bytes kann dann σ mittels einer Lookup Table direkt ausgelesen werden. Je nach Position des Bytes ist dann abschließend noch ein konstanter Wert zu addieren.

Der nächste Schritt während der Allokierung ist nun, in der Hash Table eine nicht leere Liste zu finden, die auf einen freien Speicherblock mit passender Größe verweist. Um sicher zu gehen, dass der freie Speicherblock auch wirklich über die gewünschte Größe verfugt, muss die Suche mit der Liste beginnen, welche dem berechneten $\sigma + 1$ entspricht. Leider kann der Fall auftreten, dass diese Liste gerade leer ist. Deshalb ist also die nicht leere Liste mit dem kleinsten korrespondierenden Bit ω zu finden, für die $\omega \geq \sigma + 1$ gilt. Die Vorgehensweise ist in Abbildung 7.6 aufgeführt. Zunächst wird σ innerhalb der gewünschten Größe berechnet und um eins erhöht. Der Wert μ repräsentiert eine Maske, die durch das σ-fache Shiften des Wertes $0xFFFFFFFF$ nach links entsteht. Die Verknüpfung von μ mit dem Flagspeicherwort der Listen über den boolschen Und Operator ergibt einen Wert ρ, in dem alle in Frage kommenden Listen durch ein gesetztes Bit markiert sind. Innerhalb

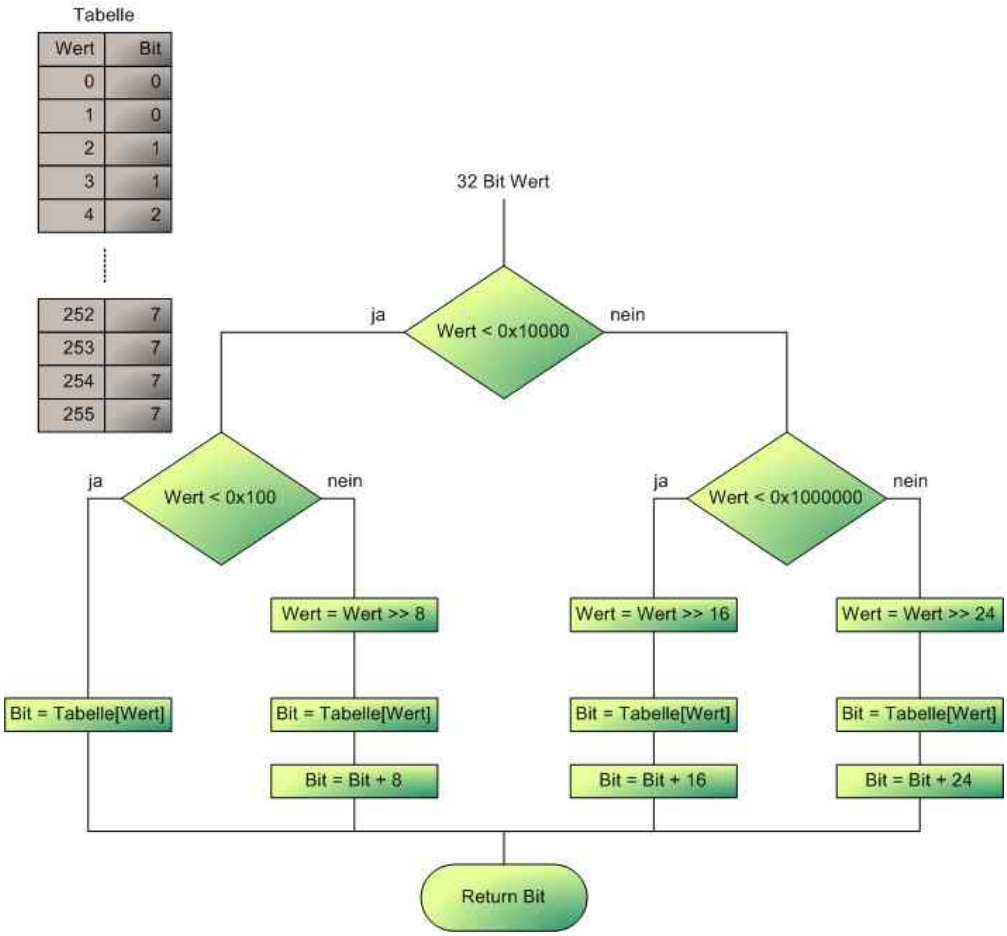

Abbildung 7.5: *Der Algorithmus zur Bestimmung der Position des höchsten gesetzten Bits innerhalb eines 32 Bit Wertes. Die Notation ist an C++ angelehnt. Das Symbol >> impliziert einen mehrmaligen Shift des Wertes nach rechts um die angegebene Zahl. Die Bezeichnung* $0x$ *leitet Hexadezimalzahlen ein.*

von ρ ist dann die Position ω des kleinsten gesetzten Bits zu finden. Der Wert ω entspricht dann dem Index der Liste innerhalb der Hash Table, d.h. Abbildung 7.6 beschreibt die Hashfunktion. Die Vorgehensweise zur Bestimmung der Position des niedrigsten gesetzten Bits erfolgt analog zur Identifikation der Position des höchsten Bits. Die Suche nach einem passenden Eintrag benötigt eine konstante Laufzeit und hat somit die Komplexität $O(n)$.

Das Verfahren entnimmt der gefundenen Liste den ersten Eintrag, reserviert dort den geforderten Speicherbereich, legt den Header des Blocks an und modifiziert die Header der benachbarten Blöcke. In der Regel ist der durch den Eintrag in der Liste repräsentierte Speicherbereich größer als der geforderte Block. Daher muss der verbleibende freie Speicher ermittelt und je nach seiner Größe als Eintrag in einer Liste der Hash Table vermerkt werden.

Beim Freigeben eines Speicherblock ist darauf zu achten, dass benachbarte Speicherblöcke möglicherweise bereits freie Blöcke darstellen. In diesem Fall sind die freien Blöcke zu einem einzigen Block zusammenzufassen und entsprechend in eine Liste der Hash Table einzuordnen.

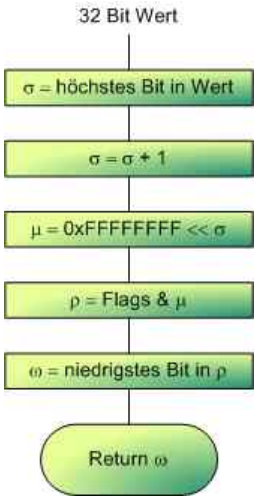

Abbildung 7.6: *Die Berechnung der Hash Funktion kommt mit einfachen boolschen sowie binären Shift Operationen aus.*

7.2.2 Key-Value Maps mit konstanten Zugriffszeiten

Aufgrund der enormen Datenmengen einer großen 3D Szene ist es auf vielen Systemen unmöglich, sämtliche Informationen im Hauptspeicher zu verwalten. Sofern ein Speichermedium wie etwa eine Festplatte vorhanden ist, muss zumindest ein Teil der Daten auf dieses Medium ausgelagert werden. Eine Information kann sich somit entweder in einer Datei oder im Hauptspeicher befinden. Mit beiden Fällen sind unterschiedliche Zugriffsweisen und somit auch entsprechende Implementierungen verbunden. Es bedeutet einen großen Mehraufwand an Programmcode und damit auch eine höhere Fehleranfälligkeit, bei jedem Zugriff stets beide Situationen zu berücksichtigen. Vielmehr sollte ein Zugriff auf eine Information für den Auftraggeber hinsichtlich Speicher oder Festplatte immer transparent verlaufen. Aus diesem Grund werden innerhalb des Systems sämtliche auslagerbare Informationen nicht über direkte Pointer sondern mittels Identifikationsnummern (ID) referenziert. Zwar ist mit letzteren die gewünschte Abstraktion möglich, jedoch fällt der Zugriff deutlich langsamer als im Falle direkter Pointer aus. Da Zugriffe beziehungsweise Suchvorgänge häufiger als Aus- und Einfügeoperationen sind, scheiden für die Assoziation zwischen ID und Daten verkettete Listen aus. Vectors oder gar Arrays wiederum sind hinsichtlich der erforderlichen Aus- und Einfügeoperationen zu ineffizient. Somit bleiben Key-Value Maps, die üblicherweise als balancierte RB-Bäume realisiert sind. Zwar verfügen RB-Bäume durchaus über ein gutes Laufzeitverhalten, ermöglichen aber im Gegensatz zu direkten Pointern keine konstanten Zugriffszeiten. Deshalb wurden im System Key-Value Maps als Template Pattern implementiert, die über genau diese Eigenschaft verfügen.

Die realisierten Maps verfügen über einen 64 Bit Adressraum, der sowohl im Hauptspeicher als auch auf der Festplatte liegen kann. Entsprechend dem Adressraum handelt es sich bei den Schlüsseln um 64 Bit Werte. Wie in Abbildung 7.7 dargestellt, entspricht die implementierte Key-Value Map ebenfalls einem Baum. Jeder innere Knoten des Baumes verfügt maximal über 256 Kinder, d.h. für das Speichern eines Index auf einen bestimmten Kindknoten werden acht Bits beziehungsweise zwei Hexadezimalziffern benötigt. Die in die Map eingefügten

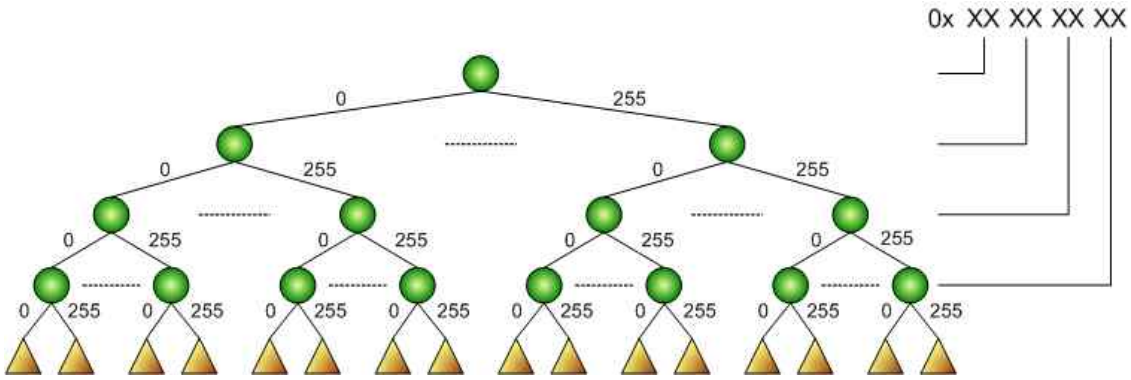

Abbildung 7.7: *Jeder Schlüssel beschreibt einen eindeutigen Pfad innerhalb des Baumes bis zu einem Blatt. Zur Vereinfachung handelt es sich hier lediglich um einen 32 Bit breiten Schlüssel, wodurch sich die Tiefe des Baumes auf 4 reduziert.*

Werte sind in den Blättern des Baumes abgelegt. Ein Pfad von der Wurzel zu einem Blatt hat immer die Länge 8, was gleichzeitig der maximalen Tiefe des Baumes entspricht. Die konstante Länge der Pfade ist letztlich auch für die konstante Suchzeit ausschlaggebend.

Im Gegensatz zu den üblichen RB-Bäumen werden die Schlüssel nicht durch den Benutzer vorgegeben, sondern beim Einfügen eines Wertes in die Map als Ergebnis zurückgeliefert. Hierzu wird der während des Einfügen traversierte Weg im Schlüssel kodiert, weshalb ein Schlüssel stets einen eindeutigen Pfad von der Wurzel bis zum jeweiligen Blatt beschreibt. Jedes Level des Baumes belegt innerhalb des Schlüssels acht Bits, wobei die Wurzel die höchstwertigsten Bits beansprucht. Beim Besuch eines inneren Knotens wird in den Schlüssel je nach Level der Index des folgenden Kindknoten eingetragen. Durch die Bitbreite des Index und die konstante Pfadlänge von 8 ergeben sich die 64 Bit eines Schlüssels. Für den Zugriff auf einen Wert unter Vorgabe des Schlüssels kann dann direkt der zu traversierende Weg ausgelesen werden.

Abbildung 7.7 suggeriert den Eindruck eines voll besetzten Baumes, was in der Praxis aber nicht der Fall ist. Ein innerer Knoten wird immer nur bei entsprechendem Bedarf angelegt, d.h. zu Beginn existiert nur die Wurzel des Baumes. Beim Einfügen des ersten Wertes werden dann die benötigten übrigen sieben inneren Knoten dynamisch angelegt. Diese Anzahl von acht inneren Knoten reicht bis zum Eintrag des 256. Wertes, anschließend ist dann ein weiterer innerer Knoten erforderlich. Abgesehen von der Wurzel kann ein innerer Knoten immer dann wieder gelöscht werden, wenn er über keine Kindknoten mehr verfügt.

Durch das Entfernen von Werten können innerhalb des Baumes freie Blätter entstehen. Aus diesem Grund beinhaltet nach Abbildung 7.8 jeder Knoten einen aufsteigend sortierten Stack mit maximal 256 Einträgen, welcher die Indices zu freien Kindknoten aufnimmt. Ein freier Kindknoten ist ein Knoten, der auf einem Pfad zu einem nicht belegten Blatt des Baumes liegt. Durch die Sortierung der Stacks ist es beim Einfügen eines Wertes nicht nur möglich, überhaupt ein freies Blatt zu ermitteln, sondern sogar effizient dasjenige freie Blatt zu identifizieren, welches dem kleinsten Schlüssel entspricht. Auf diese Weise bleibt die Map hinsichtlich des Speicherraumes kompakt, was allerdings auf Kosten der erhöhten Laufzeit während des Einfügen geht. Diese resultiert aus der Pflege der zu sortierenden Stacks.

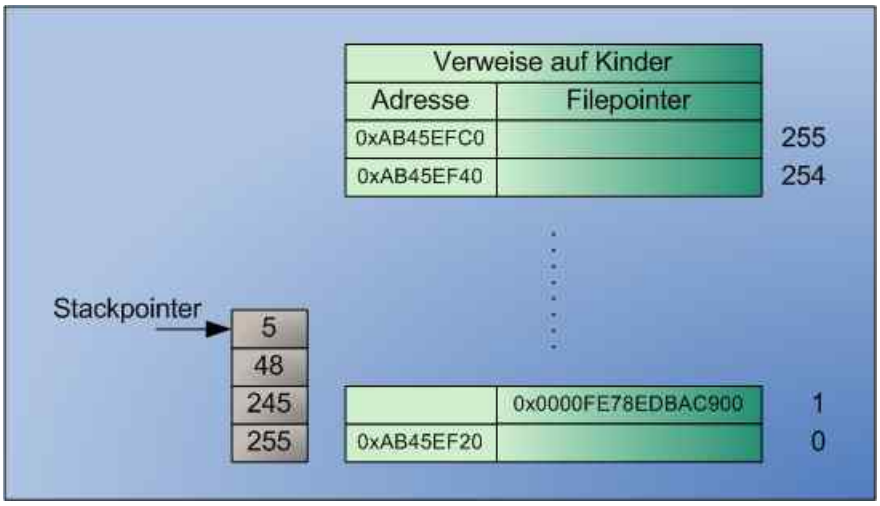

Abbildung 7.8: *Die Struktur der inneren Knoten der Map. Jeder Eintrag innerhalb des Arrays mit den Verweisen auf Kinder kann entweder eine Position innerhalb einer Datei oder eine Adresse im Speicher referenzieren. Dateipositionen sind über einen 64 Bit Wert kodiert.*

Aufgrund des 64 Bit breiten Adressraumes der Map kann bei entsprechender Anzahl der Werte selbst die innere Struktur der Map den Speicher eines Endgerätes überschreiten. Daher ist es möglich, sowohl die inneren Knoten als auch die Blätter der Map auszulagern. Für die wertespezifischen Blätter muss der Benutzer hierzu eine Schnittstelle implementieren, welches das Lesen und Schreiben eines Wertes ermöglicht. Bei der Initialisierung der Map gibt der Benutzer an, wie viel Speicher jeweils den inneren Knoten beziehungsweise Blättern zur Verfügung stehen soll. Das Aus- und Einlagern der Knoten erfolgt nach dem Verdrängungsprinzip, d.h. der am längsten nicht mehr besuchte Knoten fällt der Auslagerung zuerst zum Opfer.

Trotz der konstanten Suchzeit ist der Zugriff auf einen Wert immer noch teuer, da acht innere Knoten traversiert werden müssen. Aus diesem Grund verfügt die Map über einen assoziativen Cache, dessen Zahl an Einträgen einer 2er Potenz entspricht. Diese Anzahl ist natürlich deutlich niedriger als der mögliche 64 Bit Adressraum der Map. Somit kann es zu den von Caches bekannten Konflikten mehrerer potentieller Einträge kommen. Der Cache beinhaltet Pointer auf im Hauptspeicher befindliche Blätter der Map. Zudem ist mit jedem Pointer der korrespondierende Schlüssel vermerkt. Bei Konflikten verweist der Cache immer auf das zuletzt besuchte Blatt. Die Assoziation zwischen Schlüssel und Wert ist durch die erwähnte 2er Potenz der Cachebreite mittels einer einfachen boolschen UND Verknüpfung möglich. In Abhängigkeit der Cachebreite werden die höchstwertigsten Bits des Schlüssels abgeschnitten und der verbleibende Wert als Index an den Cache angelegt. Im Falle eines Treffers sind der vorgegebene Schlüssel und der zum Pointer vermerkte Schlüssel identisch. Andernfalls beginnt die Suche innerhalb der Map.

Der Zugriff auf die Map erfolgt über die fünf Methoden *add*, *remove*, *acquireRead*, *acquireWrite* und *release*. Die beiden ersten Methoden sorgen für das Ein- beziehungsweise Ausfügen eines Datensatzes. Die Methode *add* liefert den Schlüssel zu den gespeicherten Daten zurück. Die *acquire* und *release* Methoden schützen vor zeitgleichen Zugriffen durch mehrere Threads. Paralleles Lesen der Daten ist mittels der *acquireRead* Methode möglich. Die *acquireWrite*

Methode impliziert eine Modifikation der Daten und ist daher nicht parallel ausführbar. Ein eingelagerter Datensatz, der mindestens einmal über die *acquireWrite* Methode angefordert wurde, wird im Falle seiner erneuten Auslagerung wieder in die Datei geschrieben. Bei einem Datensatz, welcher lediglich gelesen wurde, ist das nicht erforderlich. Die *release* Methode gibt einen Datensatz wieder frei. Ein Datensatz kann nur dann ausgelagert werden, wenn er momentan nicht akquiriert ist.

7.2.3 Auslagern von Daten

Das Auslagern von Informationen in Dateien ist mitunter mit erheblichen Problemen behaftet, die oftmals von den zugrunde liegenden Betriebssystemen herrühren. Beispielsweise gibt es unter Windows für Dateien mit einer Größe über vier Gigabyte keine Möglichkeit, Daten aus einer Datei zu löschen. Dies ist aber unabdingbar, um beim permanenten Auslagern von Informationen ein Aufblähen der Dateien zu verhindern. Sofern in eine Datei Datensätze mit konstanter Länge ausgelagert werden, greift das in Abbildung 7.9 illustrierte Prinzip.

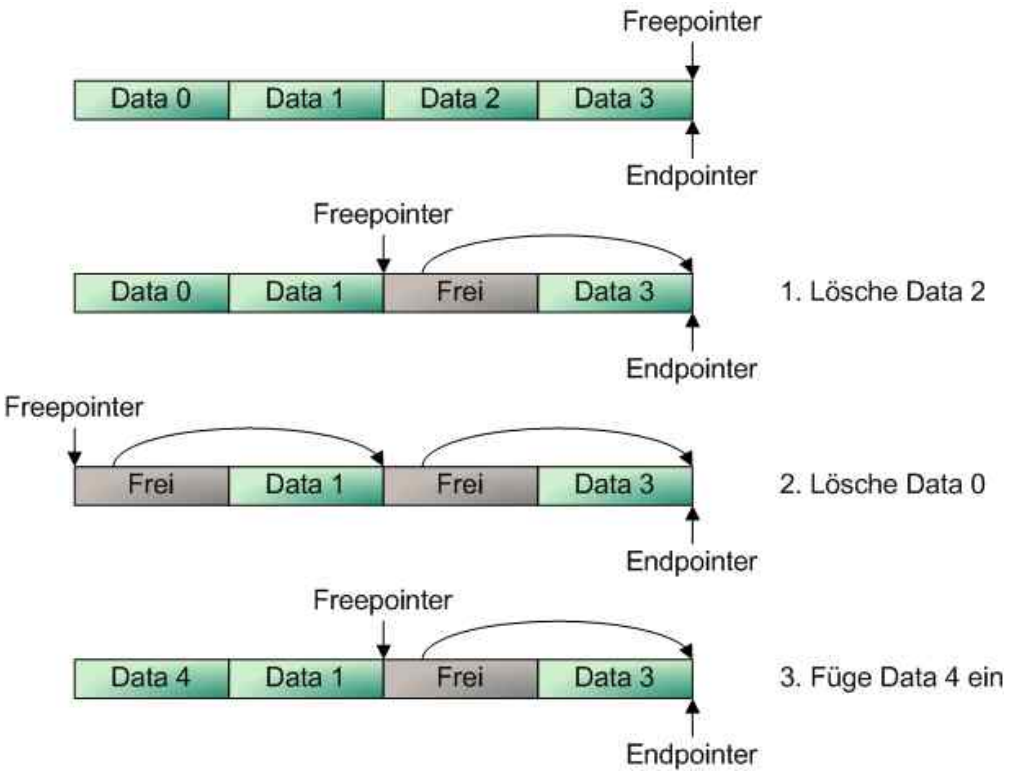

Abbildung 7.9: *Sofern die auszulagernden Daten über eine konstante Speichergröße verfügen, ist das Löschen und Einfügen von Blöcken hinsichtlich Platz- und Zeitbedarf einfach zu realisieren.*

Mit jeder Datei sind zwei Zeiger assoziiert. Der *Endpointer* verweist auf das aktuelle Ende der Datei. Der *Freepointer* zeigt auf einen freien Block innerhalb der Datei. Im einfachsten Fall verweisen beide Zeiger auf die gleiche Position, d.h. es existiert kein freier Block und die Daten müssen an das Ende der Datei angefügt werden. Beim Löschen eines Datensatzes entsteht ein freier Block, weshalb der *Freepointer* auf die Position des neuen freien Blockes

gesetzt wird. In dem freien Block wird die alte Position des *Freepointers* vermerkt. Wie ab dem zweiten Schritt in Abbildung 7.9 zu erkennen, entsteht auf diese Weise innerhalb der Datei eine verkettete Liste freier Blöcke. Verweisen *Endpointer* und *Freepointer* beim Einfügen eines Datensatzes nicht auf die gleiche Position, so erfolgt eines Speicherung des Datensatzes in demjenigen Block, auf welchen der *Freepointer* momentan verweist. Zuvor wird allerdings die im freien Block vermerkte alte Position des *Freepointers* gelesen und der Zeiger auf den entsprechenden Wert gesetzt.

Aufgrund der einfachen Handhabung von Informationen mit konstanter Länge ist ein wichtiger Ansatz innerhalb des Systems, Objekte mit unterschiedlichen Speichergrößen in mehrere Datensätze von konstanter Länge aufzubrechen und diese in getrennten Dateien abzulegen. Sofern auch diese Vorgehensweise zu aufwendig ist, wird der in Abschnitt 7.2.1 erläuterte Algorithmus analog auf Dateien angewendet.

7.3 Implementierung des Szenegraphen

Die Realisierung des Szenegraphen ist eng mit der Speicherverwaltung verbunden, da die Repräsentation der Szene den größten Teil des Speicherraumes beansprucht. Zudem kommt es hier aufgrund der möglichen Dynamik der Szenen und dem begrenzten Hauptspeicher der Systeme permanent zu Ein- und Auslagerungen verschiedener Informationen. Daten, die sich sowohl im Hauptspeicher als auch in einer Datei befinden können, werden immer über eine ID referenziert. Die Verwendung der ID anstelle direkter Adressen bietet die Möglichkeit zur Abstraktion der momentanen Lage eines Datenobjekts. Abschnitt 7.2.1 geht unter anderem darauf ein, wie der Zugriff über eine ID effizient gestaltet werden kann.

7.3.1 Implementierung des Elementgraphen

Im Unterschied zu anderen Szenegraphen wie etwa VRML beinhaltet der Elementgraph nahezu keine elementspezifischen Daten, sondern verweist lediglich auf entsprechende Einträge der Pools. Abbildung 7.10 gibt einen Überblick auf die Realisierung des Elementgraphen.

Alle Knoten des Elementgraphen erben von einer gemeinsamen Superklasse. Diese stellt die Schnittstelle zu den Renderern bereit, welche die Elementgraphen beispielsweise für eine Visualisierung traversieren. Obgleich dies nicht dem ursprünglichen objektorientierten Prinzipien entspricht, beinhalten die Knoten also keine eigenen Methoden für eine Visualisierung oder sonstige Auswertung. Stattdessen ist diese Funktionalität vollständig in den Renderern gekapselt, die für jeden Knoten eine korrespondierende Methode bereitzustellen haben. Diese Vorgehensweise hat mehrere Vorteile.

1. Die Knoten des Elementgraphen werden nicht mit unzähligen Methoden überladen. Dabei ist zu bedenken, dass je nach Betriebssystem, Graphikbibliothek oder Anwendung unterschiedliche Methoden zu implementieren sind. Die Verwendung der Renderer ermöglicht somit die Anpassung an eine bestimmte Plattform und verhindert gleichzeitig die Abhängigkeit von selbiger.

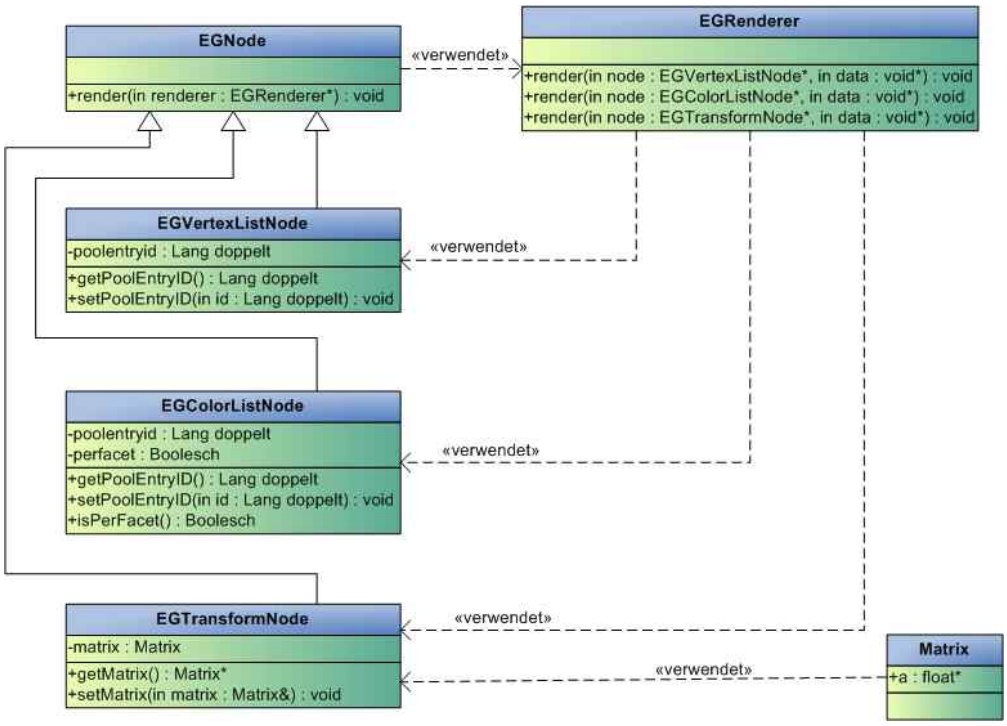

Abbildung 7.10: *Alle Knoten des Elementgraphen erben von einer gemeinsamen Superklasse.*

2. Jeder Renderer repräsentiert eine spezifische Aufgabe. Dabei kann es sich beispielsweise um die Visualisierung des durch den Graphen definierten Elements handeln. Eine andere Aufgabe wäre die Traversierung des Elementgraphen, um die AABB des Elements zu ermitteln. Der zu einer Aufgabe implementierte Code wird durch den Renderer gekapselt. Dadurch steigt die Übersichtlichkeit und Pflege der Software. Bei eventuellen Erweiterungen müssen keine bestehenden Knoten des Elementgraphen modifiziert sondern lediglich die Renderer angepasst werden.

3. Die *render* Methode der Superklasse wird von den Knoten geerbt und dort überschrieben. In jeder Knotenklasse existiert somit eine *render* Methode, welche innerhalb des als Parameter übergebenen Renderer die zum aktuellen Knoten korrespondierende Methode aufruft. Auf diese Weise wird während der Traversierung eines Elementgraphen im Renderer ein aufwändiges *case* oder *if* Konstrukt vermieden, um zunächst den Typ eines Knoten zu bestimmen und dann die entsprechende Methode aufzurufen. Abschnitt 7.4 geht noch genauer auf die Implementierung der Renderer ein.

Innerhalb des Elementgraphen gibt es zwei grundlegende Knotentypen. Knoten wie der *EGTransformNode* beinhalten direkt die Informationen, während andere Knoten wie der *EGVertexListNode* auf Einträge der Pools verweisen. Die Verwendung der Pools lohnt sich erst ab einer bestimmten Größe und Redundanz der Daten, was im Falle der Matrizen nicht gegeben ist. Einige Knotentypen wie der *EGColorListNode* verweisen auf Farbtabellen, die sowohl per Vertex als auch per Polygon definiert sein können. Zu den meisten Knoten mit Wertetabellen existiert ein progressiver Partnerknoten. Durch die Verweise auf Pooleinträge sind die Elementgraphen sehr kompakt und bedürfen wenig Speicher.

7.3.2 Implementierung der Pools

Die zentrale Anlaufstelle innerhalb der Implementierung des Pools ist der *PoolManager*. Dieser verwaltet die einzelnen Pooleinträge mittels der in Abschnitt 7.2.2 beschriebenen Key-Value Map. Wie in Abbildung 7.11 illustriert deckt sich die Schnittstelle des *PoolManagers* mit derjenigen der Key-Value-Map. Hierdurch sind parallele Zugriffe, eine Trennung zwischen lesenden und schreibenden Zugriffen sowie das Aus- und Einlagern der Pooleinträge gewährleistet. Der *PoolManager* verwendet die Superklasse *PoolEntry* aller Pooleinträge. Dort findet sich in Form der *read* und *write* Methoden die Schnittstelle zum Lesen und Schreiben eines Pooleintrages aus beziehungsweise in eine Datei. Die beiden Methoden werden von den spezifischen Ausprägungen der Pooleinträge wie etwa dem *VertexListPoolEntry* geerbt und dort überschrieben. Entsprechend der englischen Bezeichnung verwaltet die Klasse *VertexListPoolEntry* eine Liste von Scheitelpunkten. Sie steht hier exemplarisch für eine ganze Reihe weiterer Klassen, die andere Informationen wie etwa Farben, Normalen oder Texturen verwalten. In den konkreten Klassen der Pooleinträge befinden sich üblicherweise einige spezifische Methoden, die beispielsweise das Setzen oder Lesen der Scheitelpunktliste erlauben.

Wie in Abbildung 7.11 dargestellt kommt bei der Implementierung der Pooleinträge multiple Vererbung zum Einsatz. Hierfür gibt es zwei Gründe: Zum einen fällt die Implementierung eines Pooleintrages auf Client und Server unterschiedlich aus. Zum anderen gibt es zwischen der Realisierung der einzelnen Pooleinträge viele Redundanzen, weshalb sich die Auslagerung der davon betroffenen Methoden und Felder in eine spezielle Klasse lohnt. Ein Beispiel hierfür ist der *ListPoolEntry*, welcher die Verwaltung einer Liste von Informationen gleichen Typs ermöglicht. Ob es sich dabei um Farben, Normalen oder Scheitelpunkte handelt, ist letztlich egal. Die Struktur der Klasse *ListPoolEntry* ist an OpenGL angelehnt. Darunter ist keine Abhängigkeit von OpenGL zu verstehen, sondern lediglich, dass sich die Repräsentation der Datenfelder im Sinne von OpenGL bewährt hat und somit eine Nachahmung des Konzepts Sinn ergibt. Die Klasse *VertexListPoolEntry* erbt von der Klasse *PoolEntry* und von der Klasse *ListPoolEntry*. Sie ist somit in der Lage, ein Datenfeld zu verwalten und verfügt darüber hinaus über die Schnittstelle zum *PoolRender*. Dessen Einbindung ist analog zu dem in Abschnitt 7.3.1 erläuterten Prinzip der *ElementGraphRenderer*. Ein Beispiel für einen *PoolRenderer* ist unter anderem der in Abschnitt 6.6 vorgestellte Multiplexer, der einen Pooleintrag vor der Übertragung mit Hilfe der Codecs kodiert.

Die Funktionalität der Klasse *VertexListPoolEntry* ist auf Server und Client unterschiedlich. So muss der Server beispielsweise wissen, ob ein Pooleintrag bereits zu einem bestimmten Client übertragen wurde oder nicht. Daher beinhaltet der *ServerPoolEntry* für jeden registrierten Client einen *ClientEntry* mit dem entsprechenden Transferstatus. Wie oben angegeben verwaltet der *PoolManager* die Pooleinträge mittels einer Key-Value Map. Diese hat nach Abschnitt 7.2.2 die Eigenschaft, den Schlüssel eines eingefügten Wertes selbst vorzugeben. Da aufgrund der unterschiedlichen Leistungsmerkmale die Pools auf Client und Server nicht identisch sind, erhalten somit zwei zueinander korrespondierende Pooleinträge auf Client und Server unterschiedliche IDs. Hieraus ergibt sich ein Problem bei der Kommunikation zwischen Client und Server hinsichtlich des Pooleintrages. Aus diesem Grund verfügt der *ClientPoolEntry* über eine ID, welche der Identifikation auf Seiten des Servers entspricht. Der

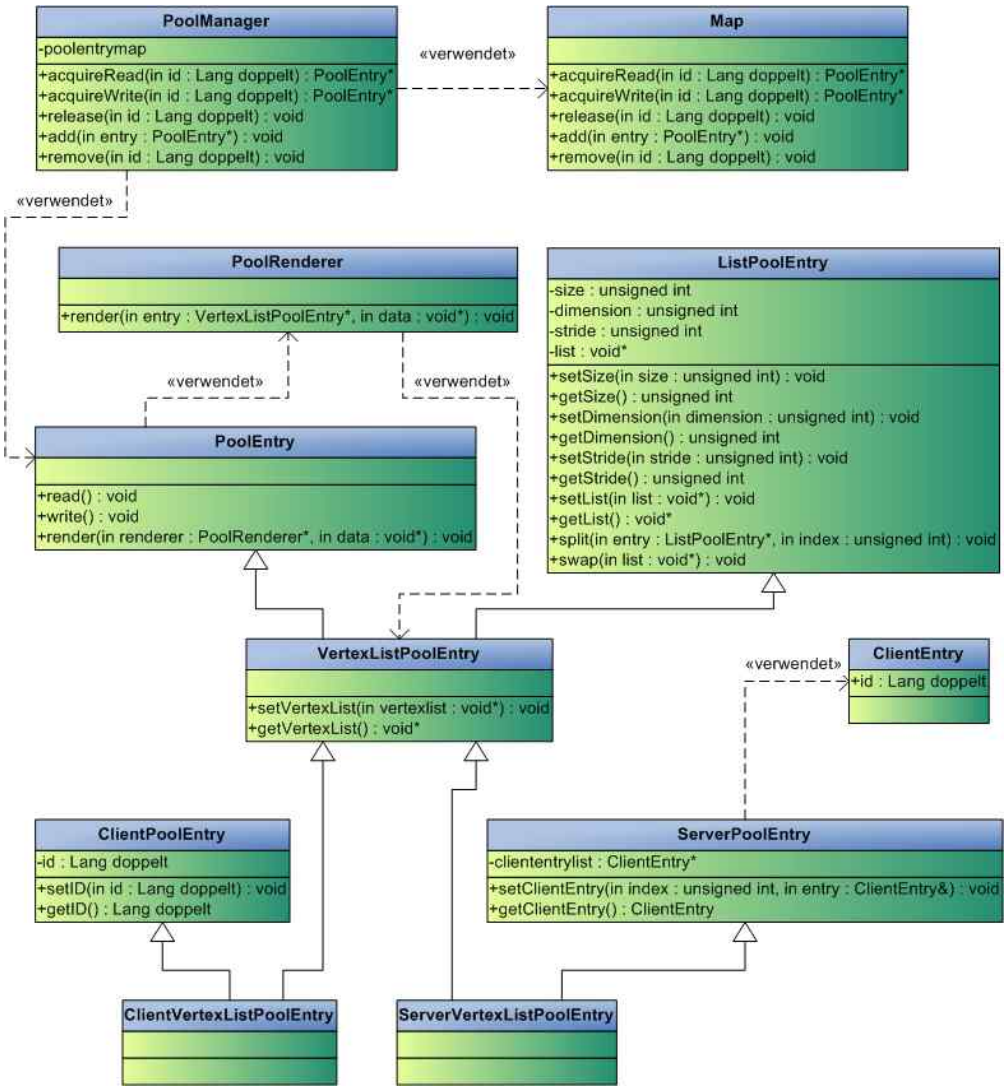

Abbildung 7.11: *Die grundlegende Klassenstruktur der Pools.*

Server dagegen verwaltet für jeden Clients mittels des *ClientEntry* die ID des Pooleintrages auf Seite des jeweiligen Clients. Dies bedeutet für den Server durchaus einen beachtlichen zusätzlichen Speicheraufwand. Daher verwendet der Server die ID zusätzlich als Flag, ob ein Eintrag bereits an einen Client übertragen wurde, d.h. nur ein gesendeter Eintrag erhält beim Server auch eine ID. Die Klassen *ClientVertexListPoolEntry* und *ServerVertexListPoolEntry* repräsentieren die endgültigen Ausprägungen des jeweiligen Pooleintrages und kommen entsprechend auf Client beziehungsweise Server zur Anwendung.

In Abbildung 7.11 entsteht die Frage, warum die Klassen *ListPoolEntry*, *ClientPoolEntry* und *ServerPoolEntry* nicht ebenfalls von *PoolEntry* erben, also keine echte, gemeinsame Superklasse existiert. Bei dieser Vorgehensweise würden in der Vererbungshierarchien sogenannte Diamanten entstehen. Entsprechende Strukturen erschweren Typecasts sowie das Erzeugen von Instanzen der Klassen mittels des Factory Patterns. Letzteres ist zum Beispiel auf Seite des Clients notwendig, um einen empfangenen Pooleintrag anhand einer Typinformation zu

identifizieren und zur Laufzeit zu erzeugen.

7.3.3 Implementierung der räumlichen Struktur

Die Schnittstelle des Szenegraphen ist durch die vier grundlegenden Methoden *add*, *remove*, *move* und *reorganize* gegeben. Alle diese Methoden können Auswirkungen auf die räumliche Struktur des Szenegraphen haben. Während die *add* Methode ein Element in den Szenegraphen einfügt, entfernt die *remove* Methode ein Element wieder aus dem Szenegraphen. Die *move* Methode verändert die Position eines Elements innerhalb der Szene. Mittels der *reorganize* Methode wird die Struktur des Szenegraphen in bestimmten Zeitintervallen entsprechend den dynamischen Vorgängen angepasst. Während der Reorganisation sind auf die Struktur des Szenegraphen keine Zugriffe möglich [3]. Die Struktur des Szenegraphen wird durch die inneren Knoten gebildet, deren Klassenhierarchie in Abbildung 7.12 illustriert ist. Alle Knoten haben mit der Klasse *SceneNode* eine gemeinsame Superklasse. Hier ist die ID des Knotens, seine AABB sowie die Referenz zum Vaterknoten abgelegt. Aller Verweise erfolgen über IDs, weshalb die inneren Knoten mittels der in Abschnitt 7.2.2 beschriebenen Key-Value Map verwaltet werden. Somit ist auch eine Auslagerung der räumlichen Struktur des Szenegraphen möglich, sofern die Größe einer Szene dies erforderlich macht.

Entsprechend dem in Abschnitt 6.3.4 erläuterten Konzept der Reorganisation, beinhaltet die Klasse *InnerNode* drei Listen. Die *currentlist* repräsentiert den aktuellen Zeitschritt und verweist auf alle Elemente, die sich momentan in dem Bereich des inneren Knotens befinden. Die *ingoinglist* und die *outginglist* verwalten dagegen die Elemente, welche im nächsten Zeitschritt den inneren Knoten betreten beziehungsweise verlassen. Hinsichtlich der Aus- und Einlagerung der Knoten ist mit den Listen ein großer Aufwand verbunden, zumal durch die Listen auch der konstante Speicherbedarf eines Knotens verloren geht (siehe Abschnitt 7.2.3). Aus diesem Grund wird bei der Implementierung des Systems häufig auf das gleiche Schema zurückgegriffen: Da jedes Element sich gleichzeitig nur in einer *currentlist*, einer *ingoinglist* und einer *outgoinglist* befinden kann, stellen die Elemente selbst die Listeneinträge dar. Elemente sind nach Abbildung 7.12 durch die äußeren Knoten beziehungsweise Blätter des Szenengraphen repräsentiert und entsprechen entweder einem *ElementNode* oder einem *GroupNode*. Letztere verwalten eine Animationshierarchie mehrerer *ElementNodes*. Analog zu den inneren Knoten werden auch alle Blätter in einer Key-Value Map verwaltet und können somit über eine ID adressiert werden.

Die Klasse *ElementNode* und die Klasse *GroupNode* erben von der Klasse *LeafNode*. Dort existieren sechs Felder die entweder mit dem Suffix *next* oder *previous* enden. Alle derartigen Felder entsprechen einer ID und verweisen auf den Nachfolger beziehungsweise den Vorgänger innerhalb der durch das Prefix gekennzeichneten Liste. Mittels der Verwendung der Knoten selbst als Teil der Liste ergeben sich mehrere Vorteile:

- Die Listen innerhalb der inneren Knoten müssen nicht aus- oder eingelagert werden. Stattdessen sind die Listen automatisch über die Knoten selbst im Speicher oder in

[3] Die Informationen der Elemente sind allerdings zugänglich, lediglich die räumliche Struktur wird blockiert

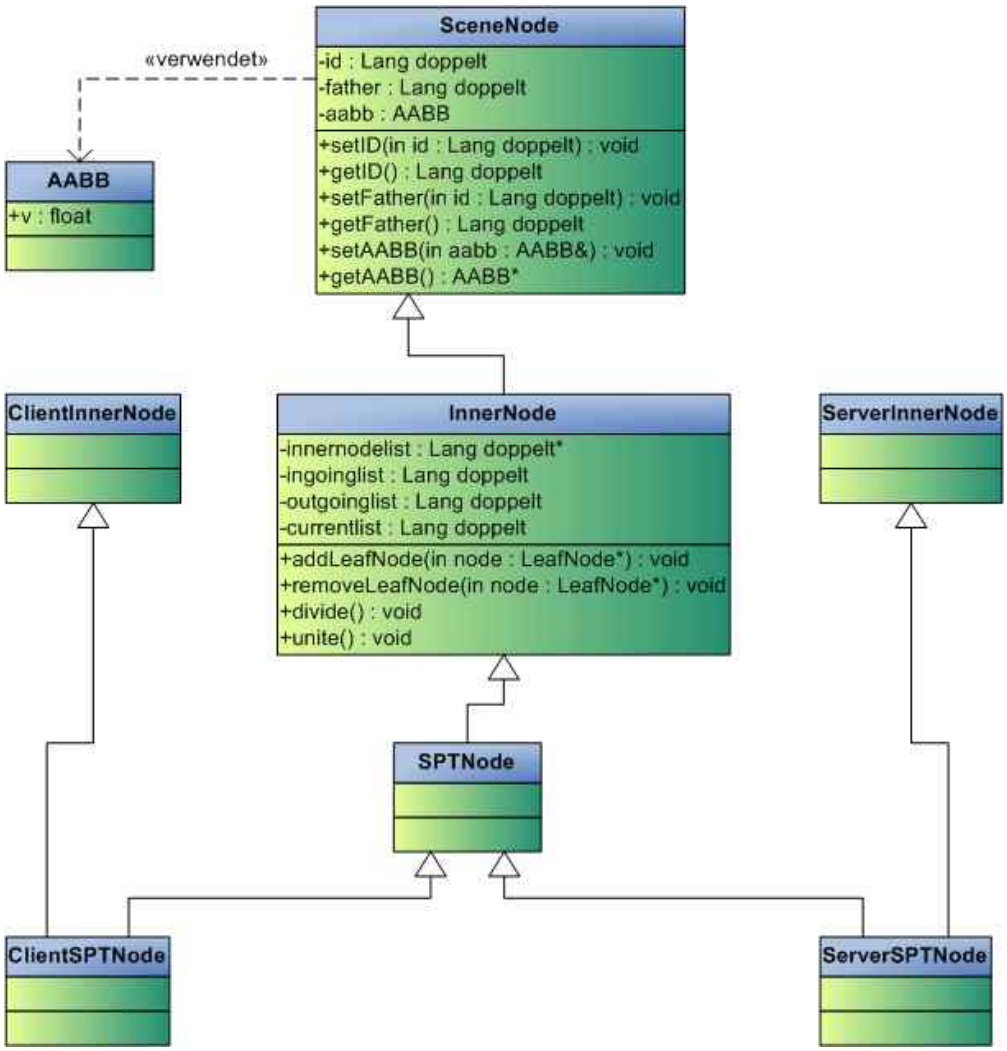

Abbildung 7.12: *Die Klassenstruktur der inneren Knoten des Szenegraphen.*

einer Datei positioniert. Nur benötigte Knoten sind eingelagert, wodurch sich eine natürliche Verwaltung der Listen ergibt.

- Die drei Listen der inneren Knoten sind jeweils nur vom Typ ID und zeigen auf das erste Element der Liste, d.h. auf den ersten *LeafNode*. Damit bleibt der konstante Speicherbedarf der inneren Knoten erhalten, was die Aus- und Einlagerung der Knoten vereinfacht.

- Mit dem Verweis auf einen Knoten ist auch gleichzeitig dessen Position innerhalb der Liste bekannt, d.h. beim Entfernen eines Knotens muss nicht erst eine Suche erfolgen.

- Aufgrund der Verwendung der ID als Verweis ist keine Aktualisierung der Listen erforderlich, sobald ein *LeafNode* aus- oder eingelagert wird.

Das gleiche Konzept findet auch bei den Animationshierarchien der *GroupNodes* Anwendung. Hier befindet sich eine *elementid*, welche auf den ersten *ElementNode* der Animationshier-

archie verweist. Die Klasse *ElementNode* wiederum beinhaltet drei dazugehörige Attribute. Dort verweist die *elementid* auf den ersten Kindknoten der Animationshierarchie. Für die Liste aller Kinder mit gleichem Level sind die Felder *hierarchynext* und *hierarchyprevious* zuständig.

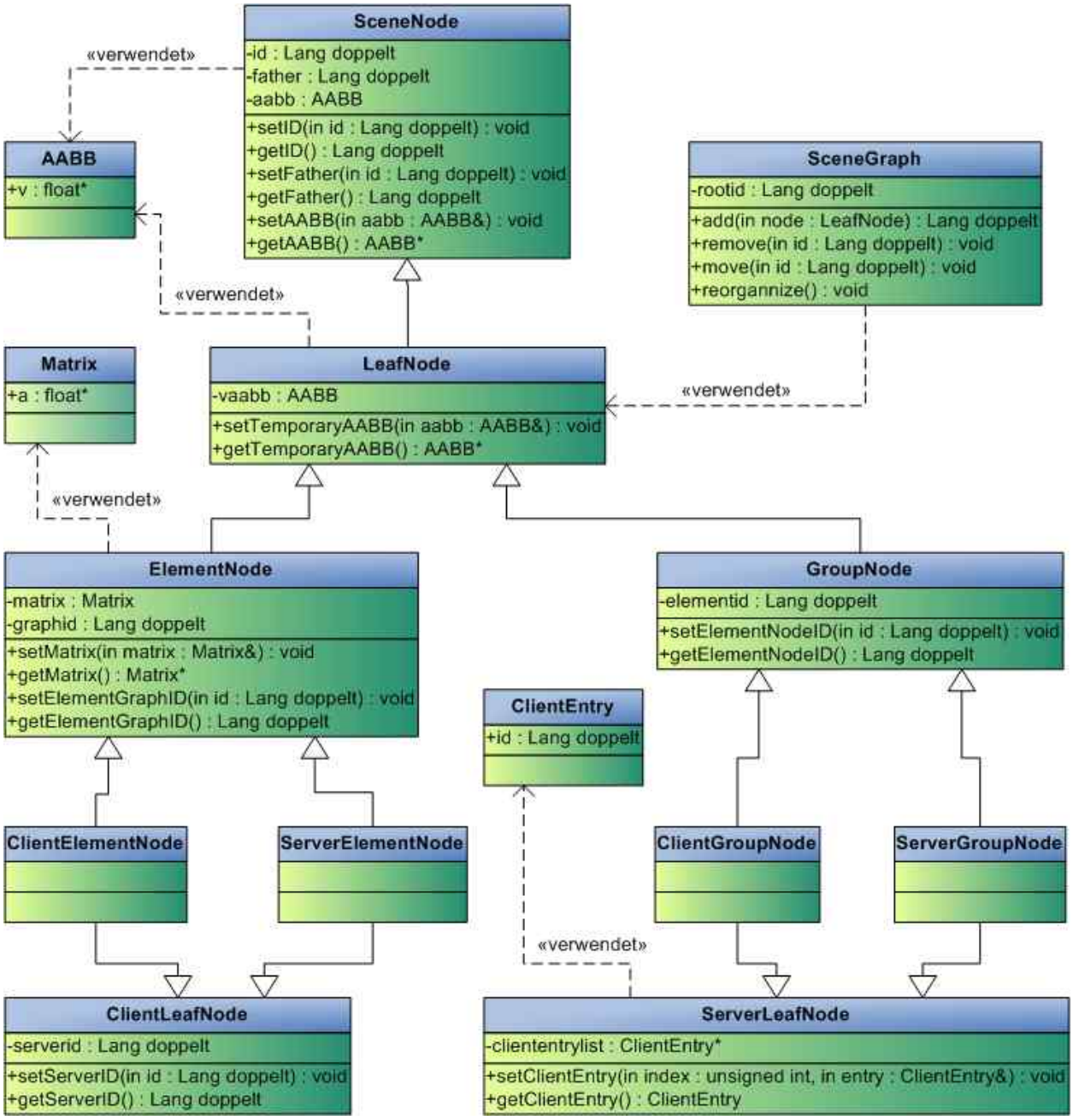

Abbildung 7.13: *Der Aufbau des Szenegraphen und die Struktur der äußeren Knoten beziehungsweise Blätter.*

Sowohl die inneren als auch die äußeren Knoten können in Abhängigkeit von Client und Server über unterschiedliche Attribute verfügen. Im Falle der äußeren Knoten greift dabei das gleiche Prinzip wie schon in Abschnitt 7.3.2 hisichtlich der Pooleinträge beschrieben. Auf Seite des Servers verfügt jedes Element über einen Array von IDs, welche die Identifikation des

Elements auf dem jeweiligen Client darstellt. Eine ID ist gleichzeitig ein Kennzeichen dafür, dass ein Element bereits zu dem betreffenden Client übertragen wurde. Aus Seite des Clients verfügt ein Element dagegen über die serverspezifische ID. Letztere und der erwähnte Array sind in der Klasse *ClientLeafNode* beziehungsweise *ServerLeafNode* implementiert. Bei den letztendlich erzeugten Instanzen handelt es sich um Objekte der Klassen *ClientElementNode*, *ServerElementNode*, *ClientGroupNode* sowie *ServerGroupNode*.

Da die hierarchische Struktur des Szenegraphen auf jedem Client eigenständig verwaltet wird und von den bislang empfangenen Elementen abhängig ist, werden die inneren Knoten nicht übertragen. Insofern fehlt hier der zuvor beschriebene Array. Bei der Realisierung der inneren Knoten ergaben sich keine client- oder serverspezifischen Merkmale. Daher sind die Klassen *ClientInnerNode* und *ServerInnerNode* lediglich aus Gründen der Symmetrie im Bezug auf die äußeren Knoten vorhanden. Außerdem besteht die Möglichkeit, dass bei einer Weiterentwicklung entsprechende Merkmale auftreten. In Abbildung 7.12 erbt die Klasse *SPTNode* von der Klasse *InnerNode*, wobei SPT für Space Partition Tree steht (siehe Abschnitt 4.4. Bei dem *SPTNode* kann es sich um eine beliebige Implementierung eines Raumunterteilungsbaumes handeln, also beispielsweise um einen Octree, eine k-D-Tree oder eben um den in Abschnitt 6.3.4 vorgestellten Baum. Die jeweilige Implementierung muss allerdings die Schnittstelle der Klasse *InnerNode* erfüllen. Dazu gehören die Methoden *divide* und *unite* sowie die Möglichkeit zum Verwalten der *LeafNodes*.

7.3.4 Berechnung der Indizes

Aufgrund der dynamischen Vorgänge können Elemente die Region eines inneren Knoten verlassen und diejenige eines anderen Knoten betreten. Ursache hierfür ist ein Aufruf entweder der *add* oder der *move* Methode. Im Falle der *add* Methode erfolgt eine Suche von oben nach unten, d.h. beginnend mit der Wurzel wird die Region eines inneren Knotens gesucht, welche die AABB des eingefügten *LeafNodes* vollständig umschließt. Beim Aufruf der *move* Methode wird dagegen zunächst von unten nach oben und anschließend von oben nach unten gesucht. Die Suche von unten nach oben endet spätestens mit der Wurzel und liefert einen übergeordneten inneren Knoten, dessen Region die transformierte AABB des Elements vollständig umschließt. Weil der gefundene Knoten oftmals nicht optimal ist, also über einen Kindknoten verfügt, dessen kleinere Region ebenfalls das Element beinhaltet, kommt es zur Suche von oben nach unten. Diese Vorgehensweise ist meistens effizienter als eine reine Suche von oben nach unten, da das transformierte Element in der Regel einen inneren Nachbarknoten betritt.

Nach Abschnitt 6.3.4 verfügt ein innerer Knoten τ über bis zu 27 Kindknoten $\tau_0 \dots \tau_2 6$, weshalb selbst die Identifikation des direkten passenden Kindknotes mit Aufwand verbunden ist. Innerhalb dynamischer Szenen sind *move* Operationen aber keine Seltenheit, sondern treten vielmehr in massiver Zahl auf. Somit ist eine effiziente Berechnung eines geeigneten Kindknoten unabdingbar. Die Zuordnung im dreidimensionalen Raum erfolgt analog zu der Zuordnung, wie sie im zweidimensionalen Fall für einen Nine Areas Tree erfolgt. Im dreidimensionalen Raum gibt es die folgenden Fälle:

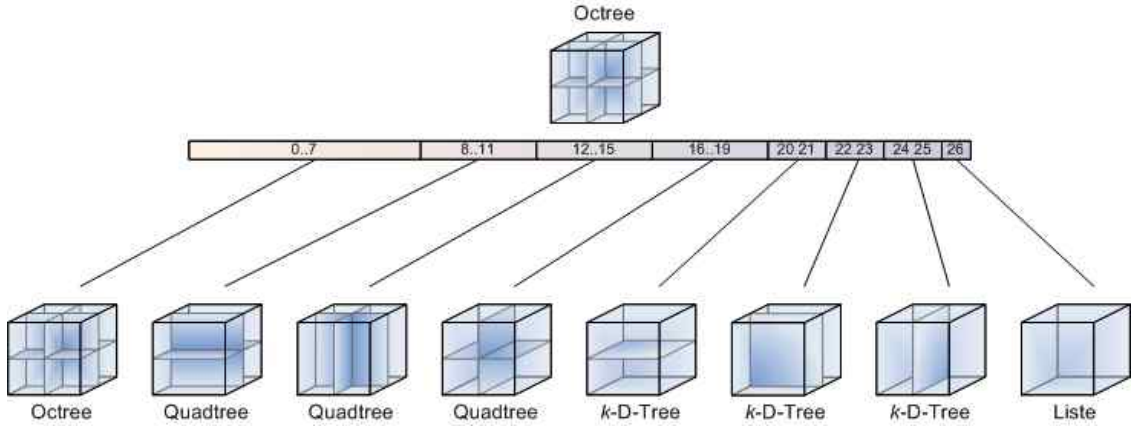

Abbildung 7.14: *Ein innerer Knoten des Raumunterteilungsbaumes hat bis zu 27 Kinder, wobei in Abhängigkeit von der Anzahl der durch ein Schnittelement geschnittenen Ebenen die Komplexität der Unterbäume abnimmt. Schneidet ein Schnittelement alle drei Ebenen, so wird ist direkt in eine lineare Liste des aktuellen inneren Knotens einsortiert.*

- Hinten / Vorne auf der z-Achse
- Links / Rechts auf der x-Achse
- Unten / Oben auf der y-Achse

Mit diesen sechs Attributen (für jede Achse zwei) lassen sich alle acht Oktanten des Octrees beschreiben. Da die einzelnen Attribute als Bits aufgefasst werden können, ergeben sich die 27 Unterräume nach Abbildung 7.14 wie folgt:

1. Im Octree wird ein Volumen mit drei Bits beschrieben. Hierbei entscheidet Bit 0 über Hinten/Vorne, Bit 1 über Links/Rechts und Bit 2 über Unten/Oben. Die entstehenden Volumen werden also mit den binären Zahlen 000 (Unten, Links, Hinten) bis 111 (Oben,Rechts,Vorne) beschrieben. Dies entspricht dem bucketnumbering Schema des Nine Areas Trees für drei Dimensionen.

2. Im Quadtree werden lediglich zwei Bits benötigt, da sich das Element auf einer Schnittebene befindet. Die betreffende Schnittebene kommt für die weitere Suche nicht mehr in Frage, weshalb nur noch die beiden verbleibenden Schnittebenen kodiert werden müssen. Die beiden Bits stehen aber nicht separat, sondern sind bedingt durch den Octree Fall zum Offset 8 aufzuaddieren. Es ergeben sich für jeden der drei potentiellen Quadtrees vier Fälle und somit für die Kindknoten die Indizes 1000-1011 (8-11), 1100-1111 (12-15) sowie 10000-10011 (16-19).

3. Der k-D-Tree Fall tritt ein, wenn ein Objekt auf 2 Schnittebenen gleichzeitig liegt. Es wird dann pro Achse nur noch ein Bit benötigt. Daraus resultieren also $3 * 2 = 6$ Möglichkeiten. Entsprechend werden die Indizes 20-21, 22-23 und 24-25 zugeordnet.

4. Für den letzten Fall gibt es keine Wahlmöglichkeit. Also wird einfach der Index 26 zugewiesen.

Die Bestimmung des Index κ des Kindknotens τ_κ, dem ein Objekt ω zugeordnet werden muss, erfolgt über die AABB von ω sowie die AABB des aktuellen Knotens τ. Aufgrund der Achsenparallelität der AABB ist deren Repräsentation als minimaler und maximaler Eckpunkt definiert, welche im folgenden als $\vec{MinBB}$ und $\vec{MaxBB}$ bezeichnet werden. Der Mittelpunkt $\tau.\vec{MidBB}$ der AABB von τ errechnet sich wie folgt:

$$\tau.\vec{MidBB} = \frac{\tau.\vec{MinBB} + \tau.\vec{MaxBB}}{2} \tag{7.1}$$

Durch den Vergleich von $\omega.\vec{MinBB}$ und $\omega.\vec{MaxBB}$ des Objektes ω mit dem Mittelpunkt $\tau.\vec{MidBB}$ ergeben sich zwei Indizes, $Index_{min}$ und $Index_{max}$. Diese beiden Indizes bezeichnen den Index des Oktanten, in welchem die Punkte $\omega.\vec{MinBB}$ beziehungsweise $\omega.\vec{MaxBB}$ liegen. Durch Vergleichen der Indizes ergibt sich dann der Index κ von τ_κ für das Objekt nach folgendem Prinzip:

- Falls $Index_{min} = Index_{max}$, dann $\kappa = Index_{min}$. Der Index des Oktanten ergibt sich sofort.
- Falls $Index_{min}$ sich von $Index_{max}$ in einem Bit unterscheidet, dann beschreibt κ den Index eines Quadtrees. κ ergibt sich abhängig davon, welche Schnittebene geschnitten wird durch $(8|12|16) + \kappa_{offset}$. κ_{offset} ist der Offset des Quadtrees und ergibt sich aus den beiden übrigen Bits.
- Falls $Index_{min}$ sich von $Index_{max}$ in zwei Bits unterscheidet, dann ist κ der Index eines k-D-Trees. κ ergibt sich abhängig von den beiden Schnittebenen durch, $(20|22|24) + \kappa_{offset}$. κ_{offset} ist hier 0 oder 1, abhängig vom Wert des gleichen Bits in $Index_{min}$ und $Index_{max}$.
- Falls $Index_{min}$ sich von $Index_{max}$ in sogar allen drei Bits unterscheidet, tritt der letzte Fall ein und der Index ist 26.

7.3.5 Implementierung der Reorganisation

Nach Abschnitt 6.3.4 liegt der Reorganisation des Szenegraphen das Konzept der kompensierenden Bewegungen zugrunde. Hierzu werden sämtliche Modifizierungen durch dynamische Vorgänge gesammelt und dann durch Aufruf der Methode *reorganize* analysiert und in die neue Struktur der räumlichen Repräsentation umgesetzt. Dieser Abschnitt beschreibt einen Algorithmus zur Umsetzung der kompensierenden Bewegungen. Ein Schwerpunkt ist dabei, möglichst wenig Zugriffe auf die einzelnen Knoten des Szenegraphen zu haben. Der Hintergrund hierfür ist die Gefahr, dass ein Knoten sich stets auf einem externen Medium befinden kann, also für einen Zugriff extra in den Hauptspeicher eingelagert und stattdessen ein anderes Element ausgelagert werden muss.

Veränderungen der räumlichen Struktur treten in Folge des Aufrufs der drei Methoden *add*, *remove* und *move* auf. Die *add* Methode fügt einem inneren Knoten ein Element hinzu,

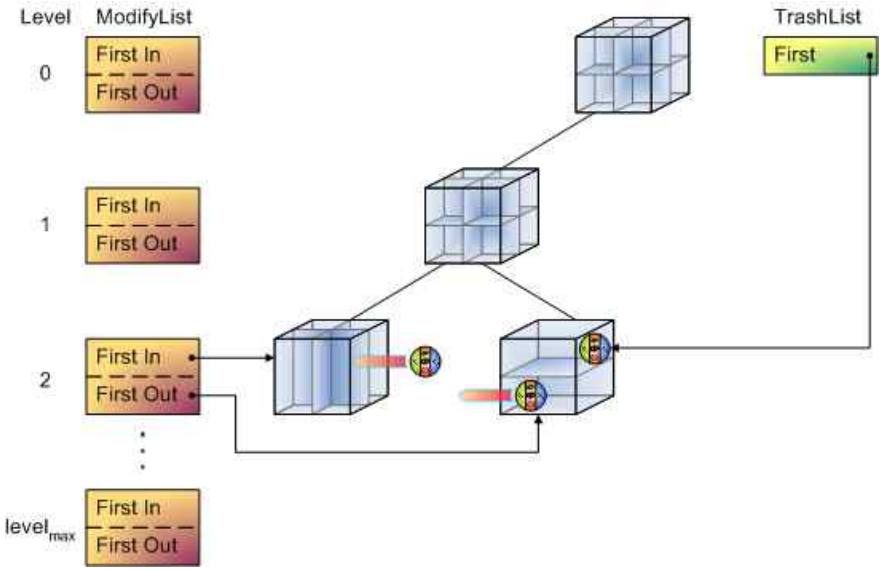

Abbildung 7.15: *Die Hilfsstrukturen zur Reorganisation des Szenegraphen.*

trägt also einen *LeafNode* in die *ingoinglist* des inneren Knotens ein. Analog verkettet die *remove* Methode einen *LeafNode* in der *outgoinglist*. Eine Besonderheit der *remove* Methode ist die Eigenschaft, die *LeafNodes* zusätzlich in der *TrashList* zu vermerken. Betroffene *LeafNodes* werden während der Reorganisation vollständig aus dem Szenegraphen entfernt. Im Gegensatz zu den beiden erstgenannten Methoden modifiziert die *move* Methode zwei innere Knoten und trägt den betroffenen *LeafNode* zum einen in die *outgoinglist* des verlassenen inneren Knotens und zum anderen in die *ingoinglist* des betretenen inneren Knotens ein. Nach Abbildung 7.15 existieren für jedes Level des Raumunterteilungsbaumes zwei Listen, markiert durch *First In* beziehungsweise *First Out*. Die maximal möglich Anzahl an Levels des Raumunterteilungsbaumes wird durch den Benutzer vorgegeben. Die aktuelle Implementierung legt die Grenze auf 64 fest, was auf der einen Seite exorbitante Szenen ermöglicht und auf der anderen Seite den Speicherbedarf hinsichtlich der Listen kaum beansprucht. Die Referenzierung der Listen erfolgt analog zu der in Abschnitt 7.3.3 beschriebenen Vorgehensweise, d.h. *First In* und *First Out* beinhalten die ID zum ersten Knoten der Liste. Die Verkettung zu weiteren Knoten erfolgt dann über die Knoten selbst. Abbildung 7.12 ist um die entsprechenden Attribute zu erweitern.

Sobald ein innerer Knoten durch die drei oben genannten Methoden modifiziert wird, kommt es zu einer Eintragung des inneren Knotens in eine Liste mit korrespondierendem Level. Ist die *ingoinglist* des inneren Knoten Liste betroffen, so wird der Knoten in der *In* Liste vermerkt. Im Falle der *outgoinglist* erfolgt entsprechend der Verweis in der *Out* Liste. Jeder innerer Knoten kann sich gleichzeitig in einer *In* und einer *Out* Liste befinden, aber jeweils nur einmal. Somit beinhalten die beiden *ModifyLists* vor Aufruf der *reorganize* Methode genau diejenigen inneren Knoten, welche eine Umstrukturierung des Raumunterteilungsbaumes erforderlich machen.

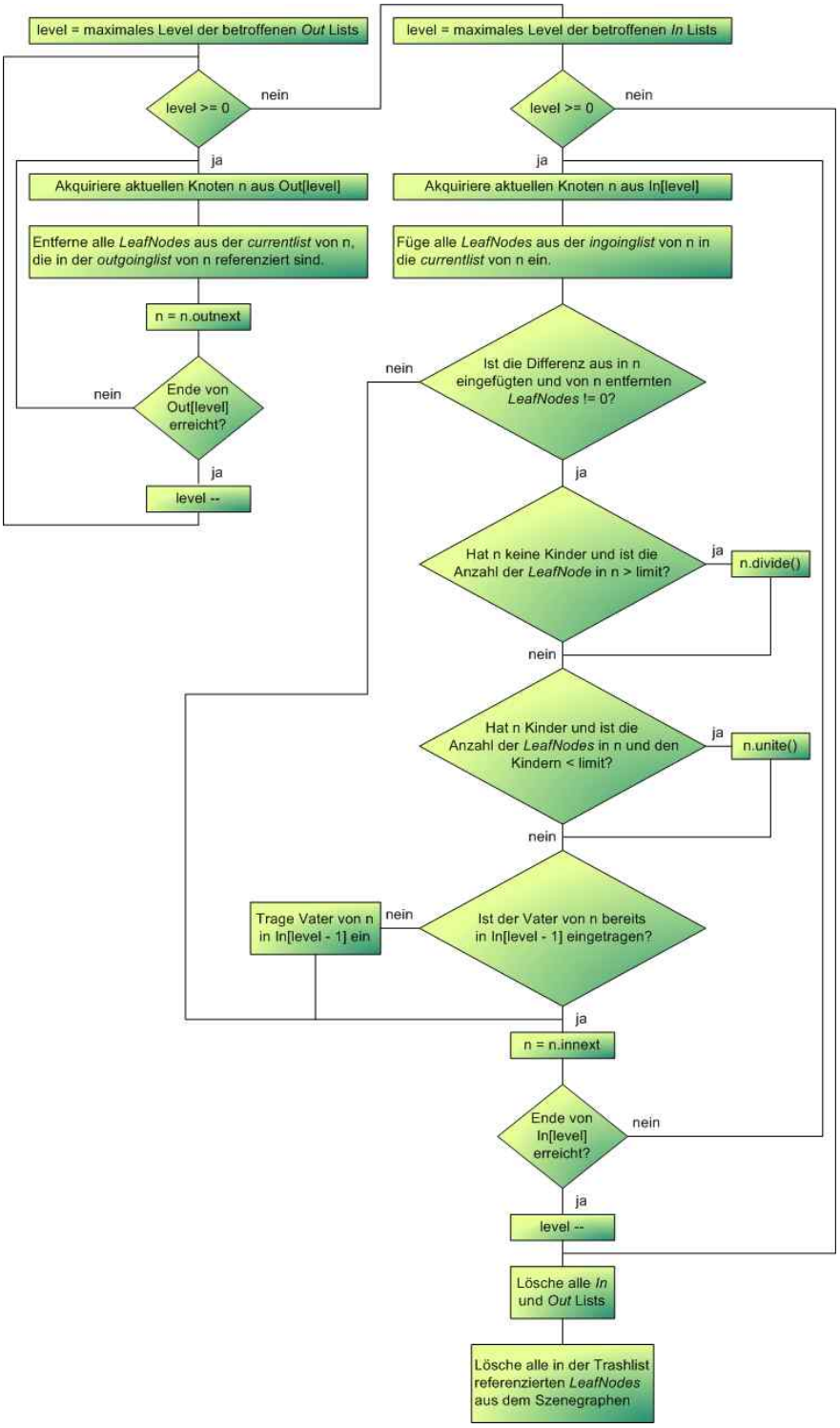

Abbildung 7.16: *Ein vereinfachtes Ablaufdiagramm zur Reorganisation des Szenegraphen unter Berücksichtigung kompensierender Bewegungen.*

Der Algorithmus zur Reorganisation bearbeitet zunächst alle inneren Knoten, die über die *Out* Listen referenziert sind, d.h. sämtliche von einem *LeafNode* verlassenen inneren Knoten. Der Grund hierfür liegt in der Struktur der Klasse *LeafNode*, wonach ein *LeafNode* sich gleichzeitig nur in einer *currentlist* eines inneren Knotens befinden kann. Somit muss ein *LeafNode* zunächst aus einem inneren Knoten entfernt werden, bevor er im Zielknoten wieder eingefügt werden kann. Genau dieser Vorgang erfolgt im ersten Abschnitt der Reorganisation. Beginnend mit dem höchsten Level, welches über eine nicht leere *Out* Liste verfügt, arbeitet der Algorithmus alle in den *Out* Listen eingetragenen inneren Knoten Level für Level ab. Dabei werden die in der *outgoinglist* referenzierten *LeafNodes* aus der *currentlist* des inneren Knotens entfernt. Anschließend wird die *outgoinglist* gelöscht.

Der zweite Abschnitt hat eine analoge Vorgehensweise, behandelt aber alle über die *In* Listen adressierten inneren Knoten. Hier werden nun alle über die *ingoinglist* vermerkten *LeafNodes* an die *currentlist* des inneren Knotens angehängt. Aus der Differenz zwischen eingefügten und entfernten *LeafNodes* ergibt sich ein Wert, welcher das Ausmaß der Dynamik im inneren Knoten wiederspiegelt. Entspricht dieser Wert Null, dann haben sich hinsichtlich der räumlichen Struktur keine Auswirkungen ergeben. Andernfalls überprüft der Algorithmus, ob der innere Knoten mittels der Methoden *divide* und *unite* unterteilt beziehungsweise zusammengefasst werden muss. Als Kriterium dient dabei die vordefinierte Kapazität, in Abbildung 7.16 als *limit* bezeichnet. Da Veränderungen innerhalb eines inneren Knotens auch dessen Vaterknoten beeinflussen können, ist auch der Vaterknoten zu überprüfen und somit in die nächst niedrige *In* Liste einzutragen. Abschließend löscht der Algorithmus alle *In* sowie *Out* Listen und entfernt die in der *TrashList* eingetragenen *LeafNodes* aus dem Szenegraphen.

7.4 Implementierung der Renderer

Auf die Implementierung der Renderer geht bereits Abschnitt 7.3.1 ein. Insgesamt existieren Renderer für den Elementgraphen, die Pools sowie den Raumunterteilungsbaum. Die grundlegende Vorgehensweise ist dabei immer die gleiche: Der Renderer beinhaltet für jeden Knotentyp beziehungsweise Pooleintrag eine korrespondierende Methode, welche den spezifischen Programmteil ausführt. Knoten und Pooleinträge weisen in ihrer Schnittstelle eine *render* Methode auf, der als Parameter der Renderer übergeben wird. Innerhalb der *render* Methode steht dann der Aufruf der korrespondierenden Methode des Renderers.

Hierarchische Strukturen wie der Elementgraph und der Raumunterteilungsbaum verfügen über Knoten, die andere als Kinder referenzieren können. In diesen Fällen erfolgt die Traversierung der Struktur innerhalb des Renderers. Beispielsweise beinhaltet ein Elementgraphrenderer eine Methode für den GroupNode, welche dessen Kinder identifiziert und der Reihe nach die *render* Methoden der Kinderknoten aufruft. Als Parameter wird der aktuelle Elementgraphrenderer verwendet.

Wie in Abbildung 7.17 dargestellt hat diese Vorgehensweise den Nachteil, zunächst einmal durch den Aufruf der *render* Methode eines Knotens den Renderer zu verlassen und dann von

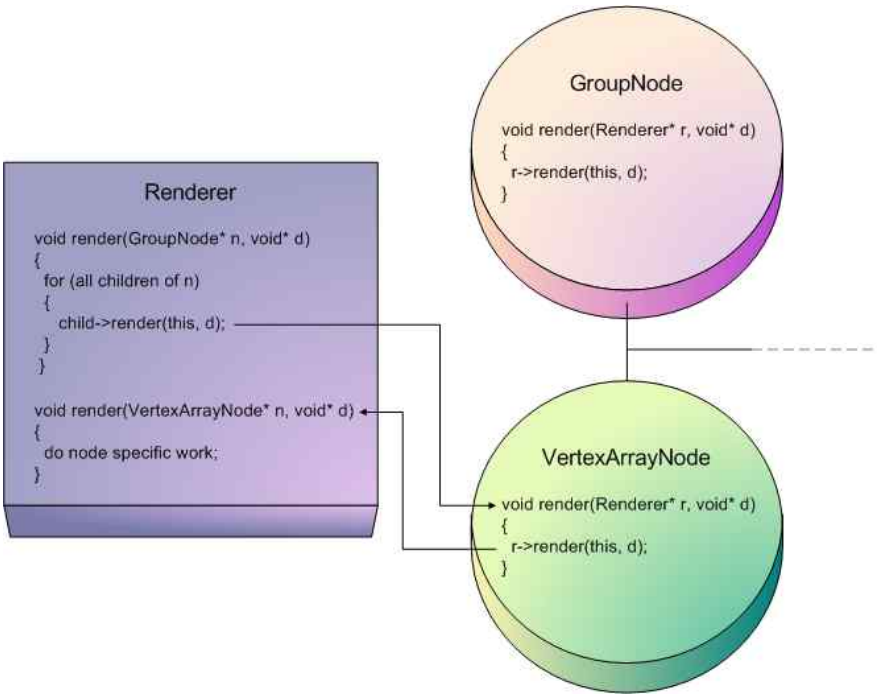

Abbildung 7.17: *Der Kontrollfluss verlässt den Renderer, um den Knoten zu identifizieren und kehrt von dort wieder zum Renderer zurück. Der zweite Parameter d erlaubt die Übergabe von Hilfsdaten.*

dort wieder in den Renderer zurückzukehren. Allerdings ist es dadurch möglich, die Funktionalität einer spezifischen Ausgabe komplett im Renderer zu kapseln. Für die realisierte Lösung gibt es zwei Alternativen: Zum einen wäre es möglich, innerhalb der Renderer aufwändige *switch* Anweisungen zu verwenden, welche den Knotentyp identifizieren und dann die korrespondierende Methode aufrufen. Zum anderen könnten auch *Function Pointer* in Kombination mit einer Sprungtabelle zum Einsatz kommen.

Beide Alternativen sind hinsichtlich der Pflege beim nachträglichen Einführen neuer Knotentypen oder Pooleinträge umständlich zu handhaben, weil der Code im Renderer an mehreren Stellen verändert werden muss [4]. Dagegen ist bei der verwendeten Lösung lediglich eine Methode in den Renderer einzufügen. Dies kann auch durch Erben des ursprünglichen Renderers geschehen, wobei der erbende Renderer dann eine weitere Methode hinzufügt. Die Komplexität der *switch* Anweisung liegt bei O(n). Der Grund hierfür ist bei der Umsetzung der Anweisung durch den Compiler zu suchen, da selbiger letztlich aufeinander folgende Bedingungen erzeugt. Im Gegensatz dazu weisen sowohl die Function Pointer als auch die realisierte Vorgehensweise konstante Laufzeiten auf, wobei erstere minimal schneller ausfallen. Wegen der besseren Pflege fiel dann die Wahl auf die in Abbildung 7.17 illustrierte Vorgehensweise.

[4] Beispielsweise in besagter *switch* Anweisung

7.5 Implementierung der progressiven Simplifizierung

Die progressive Simplifizierung schließt sowohl die Vereinfachung eines Dreiecksnetzes als auch dessen Verfeinerung ein. In beiden Prozessen kommt es zu Modifikationen der zugrunde liegenden Datenstrukturen, da Scheitelpunkte und Dreiecke entfernt beziehungsweise eingefügt werden müssen. Flexible Strukturen wie etwa Bäume oder Listen, die für massive Modifikationen geeignet sind, weisen das Problem auf, in der Regel nicht für eine schnelle Ausgabe mittels der Grafik Hardware geeignet zu sein. So verlangt beispielsweise eine performante Visualisierung mit OpenGL einen Array hintereinander liegender Informationen, der über einen einzelnen Pointer adressiert und dem Grafik Chip übergeben wird. Arrays wiederum sind für eine Vielzahl von Modifikationen nicht geeignet, da für eine konsistente Speicherdarstellung die Verschiebung großer Datenmengen erforderlich ist. Effiziente Manipulationen von Arrays sind entweder das Anfügen von Informationen an das Ende des Arrays [5], sowie das Vertauschen zweier Informationen innerhalb des Arrays. Wie in Abschnitt 6.4 erläutert bilden diese beiden Operationen die Grundlage für die progressive Simplifizierung.

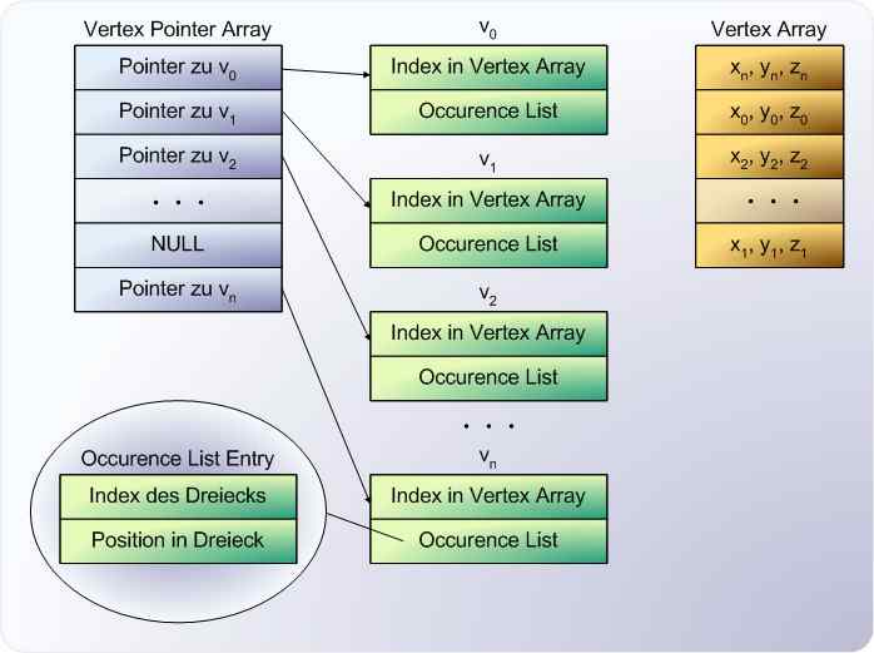

Abbildung 7.18: *Die Verwaltung der Scheitelpunkte für die Vereinfachung eines Dreiecksnetzes.*

Während der Vereinfachung des Polygonnetzes wird mittels der Metrik eine Kantenkollabierung identifiziert, welche letztlich die Entfernung eines Scheitelpunktes sowie zweier oder mehr Dreiecke zur Folge hat. Daher benötigt die Vereinfachung zusätzliche Informationen wie etwa die Angabe, in welchen Dreiecken ein bestimmter Scheitelpunkt eingetragen ist. Abbildung 7.18 illustriert die Verwaltung der Scheitelpunkte für ein Dreiecksnetz. Der *Vertex Array* beinhaltet die eigentlichen Werte der Scheitelpunkte in einer ausgabefreundlichen Art

[5] Natürlich nur dann, wenn der zu erwartende Speicherraum bereits reserviert wurde. Während im Falle der Vereinfachung der Array seine Größe beibehält, kann für die Verfeinerung der Speicherbedarf abgeschätzt werden.

und Weise, d.h. alle Daten liegen hintereinander im Speicher und weisen keine Löcher auf. Dies steht im Gegensatz zum Vertex Pointer Array, der durchaus leere Einträge verwalten kann. Der Vertex Pointer Array adressiert mit Hilfe von Pointern Informationen, die während der Vereinfachung benötigt werden. Dazu gehört der Index des Scheitelpunktes im Vertex Array, um die eigentlichen Werte des Scheitelpunktes bestimmen zu können. Eine weitere wichtige Information ist die *Occurence List*, deren Eintrag in Abbildung 7.18 links unten aufgeführt ist. Sie ermöglicht die Identifikation der von einem Scheitelpunkt betroffenen Dreiecke. Der Index des Dreiecks bezieht sich dabei auf den *Triangle Array* in Abbildung 7.19, welcher analog zum *Vertex Array* strukturiert ist. Mögliche Positionen eines Scheitelpunktes innerhalb eines Dreiecks liegen von $0 \ldots 2$. Die Vereinfachung kann also mittels des *Vertex Pointer Arrayx* Scheitelpunkte aus dem Dreiecksnetz entfernen und die betroffenen Dreiecke identifizieren. Der *Vertex Array* bleibt immer konsistent, da lediglich Informationen vertauscht werden. Mögliche per Vertex Zusatzinformationen wie etwa Normalen oder Farben sind wie der *Vertex Array* angelegt und benötigen daher zur Vereinfachung keine gesonderten Strukturen. Für die Verfeinerung ist ausschließlich der *Vertex Array* erforderlich.

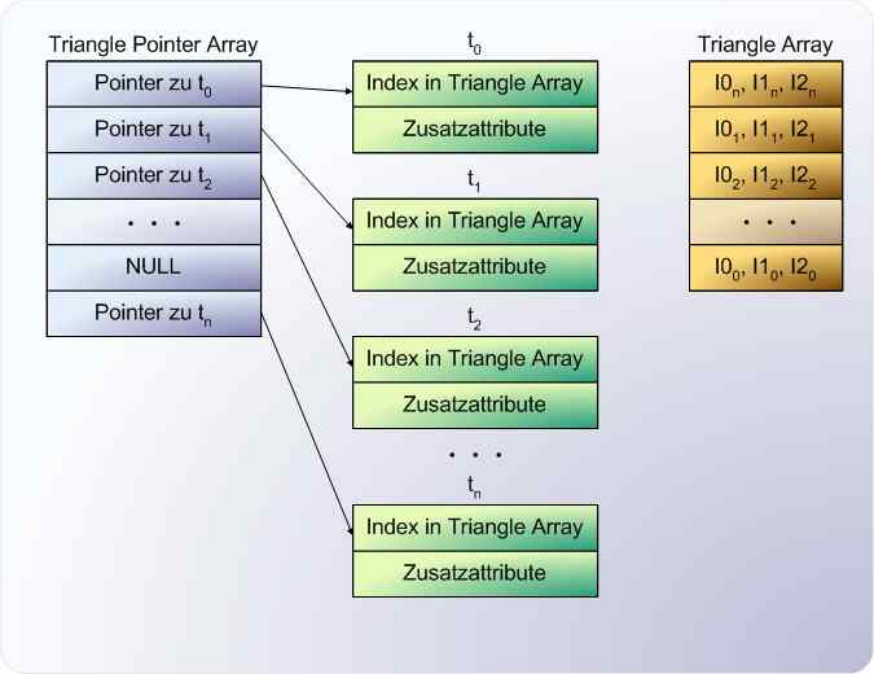

Abbildung 7.19: *Die Verwaltung der Dreiecke für die Vereinfachung eines Dreiecksnetzes.*

Die Verwaltung der Dreiecke erfolgt ähnlich zu den Scheitelpunkten. Der *Triangle Array* enthält Indizes auf Scheitelpunkte im *Vertex Array*. *Triangle Array* und *Vertex Array* zeichnen sich somit für die schnelle Ausgabe während Vereinfachung und Verfeinerung verantwortlich. Per Facet definierte Zusatzinformationen sind in den durch den *Triangle Pointer Array* adressierten Einträgen gespeichert und liegen daher nicht in einer konsistenten Speicherrepräsentation vor. Für eine schnelle OpenGL Ausgabe ist das auch nicht notwendig, da die *Interleaved Array* Funktionalität von OpenGL für per Vertex Informationen gedacht ist.

7.6 Implementierung des Schedulers

Sobald sich ein Client beim Server anmeldet, erzeugt der Server eine ID für den Client und richtet jeweils Einträge mit Informationen über den Client im Scheduler und im Multiplexer ein. Bei Annahme der Registrierung sendet der Server eine entsprechende Nachricht mit der ID als Inhalt an den Client zurück. Hat der Benutzer eine Szene ausgewählt und beginnt mit der Betrachtung und Begehung der Szene, dann sendet der Client gemäß Abschnitt 6.7.1 in regelmäßigen Abständen die aktuelle Area of Interest des Benutzers an den Client. Wie in Abschnitt 6.5 erläutert, wertet der Scheduler die Areas of Interest aller Clients aus und übergibt die zu übertragenden Elemente an den Multiplexer. Abbildung 7.20 illustriert die beschriebene Vorgehensweise.

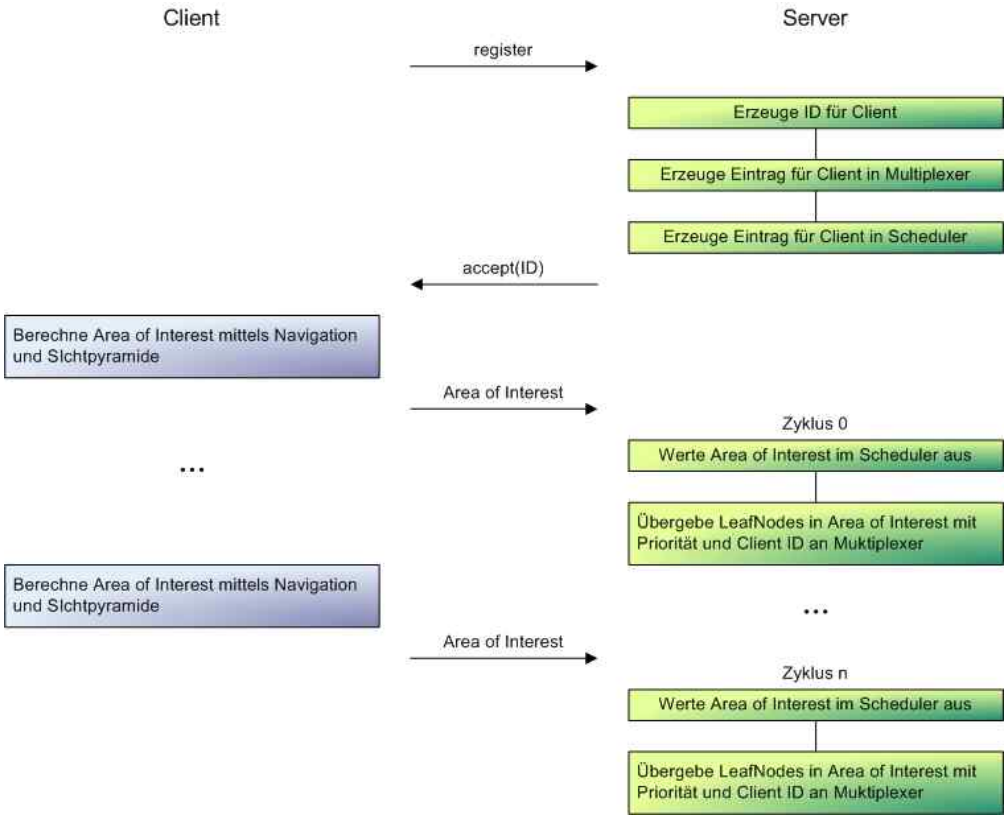

Abbildung 7.20: *Nachdem sich ein Client beim Server registriert hat, wertet der Server in regelmäßigen Abständen im Scheduler die Area of Interest aus und überträgt die betroffenen Elemente schrittweise an den Client.*

Der Scheduler ist als Renderer mit eigenem Thread beziehungsweise Kontrollfluss implementiert, welcher die räumliche Struktur des Szenegraphen traversiert. Während der Traversierung wird der Raumunterteilungsbaum gegen jeweils einen der drei Clientbäume getestet. Ein Clientbaum repräsentiert dabei die Interessenzonen aller Clients mit einer bestimmten Priorität, d.h. ein Clientbaum stellt alle sichtbaren Zonen, ein Clientbaum alle dynamischen Zonen und ein Clientbaum alle präventiven Zonen dar. Die Neuerstellung eines Clientbaumes ist immer dann erforderlich, sobald mindestens ein Client seine Area of Interest verändert. Falls notwendig erfolgt die Erzeugung des Clientbaumes stets vor dem Test gegen

den Raumunterteilungsbaum. In der aktuellen Implementierung kommt ein 1-2-4 Rhythmus zu tragen. Pro Zyklus wird der Raumunterteilungsbaum also einmal gegen den Clientbaum mit der niedrigsten Priorität getestet, zweimal gegen den mittleren Clientbaum und viermal gegen den Clientbaum mit der höchsten Priorität. Während des Testes werden die *LeafNodes* identifiziert, die sich innerhalb der Interessenzone eines Clients befinden, und die ID des betroffenen *LeafNodes* und des Clients sowie die Priorität des Clientbaumes an den Multiplexer übergeben.

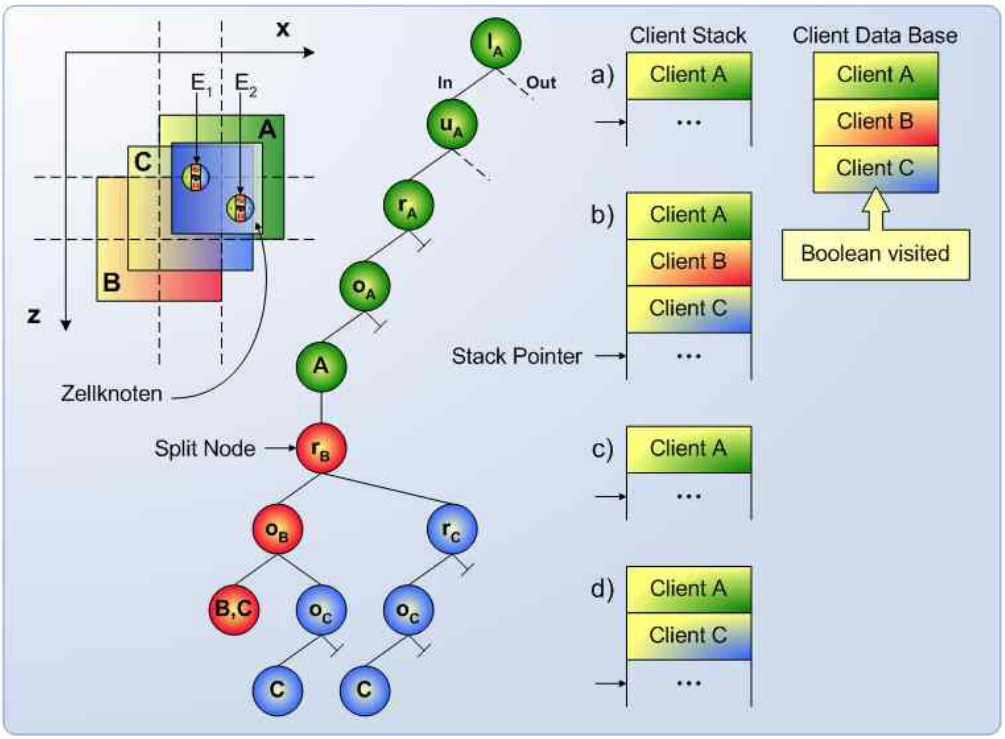

Abbildung 7.21: *Das Problem redundanter Zuweisungen wird mittels eines Stacks gelöst.*

Wie in Abschnitt 6.5 erläutert, kann ein Clientbaum durchaus mehrere Clientknoten beinhalten, welche denselben Client referenzieren. Wird also ein *LeafNode* gegen den Clientbaum getestet, so kann es prinzipiell pro Test zu mehreren Zuweisungen des *LeafNodes* an den Multiplexer kommen. Letzterer müsste dann stets den aktuellen Status der Übertragung des *LeafNodes* an den Client identifizieren, wobei das Resultat bereits im vorhinein feststeht: Der *LeafNode* ist beim Multiplexer angemeldet [6]. Es gilt daher redundante Zuweisungen des Schedulers an den Multiplexer zu vermeiden. Abbildung 7.21 gibt hierzu einen Überblick auf die realisierte Vorgehensweise.

Die *Client Data Base* ist ein Vector, der innerhalb des Schedulers die registrierten Clients verwaltet. Von Bedeutung ist dabei das *visited* Flag, welches jedem Eintrag des Vectors zueigen und mit *False* initialisiert ist. Der *Client Stack* enthält Verweise auf Einträge der *Client*

[6] Ob der *LeafNode* sich im Wartezustand oder bereits in der Übertragung befindet oder gar schon gesendet wurde ist aus Sicht des Schedulers unerheblich. Lediglich die Anmeldung des *LeafNodes* beim Multiplexer ist von Bedeutung. Eine neue Priorität, die ein *LeafNode* während seiner Übertragung erhält, kann sich erst mit einem neuen Zyklus auswirken.

Data Base und wird durch den Test des Raumunterteilungsbaumes gegen einen Clientbaum manipuliert. Abbildung 7.21 zeigt eine Situation, in der drei Clients A, B und C registriert sind. Die Interessenzonen der drei Clients überlappen sich gegenseitig und ergeben einen Clientbaum, der ausschnittsweise Abbildung 6.26 entnommen ist. Gegen diesen Clientbaum wird nun ein Zellknoten beziehungsweise innerer Knoten getestet, welcher zwei Elemente E_1 und E_2 umschließt. Bei der Traversierung durchläuft der innere Knoten den Teilbaum von A und erreicht den Clientknoten A ohne Unterteilung, da er sich komplett in der Interessenzone von A befindet. Mit dem Besuch des Clientknotens wird A auf dem Stack vermerkt und das entsprechende *visited* Flag der *Client Data Base* auf *True* gesetzt (Situation a). Anschließend ist allerdings eine Unterteilung des inneren Knotens erforderlich, weshalb die beiden Elemente E_1 und E_2 gegen den durch den *Split Node* markierten Teilbaum getestet werden müssen. E_1 wiederum benötigt eine Unterteilung beim Knoten o_B, wobei der in B und C liegende Bereich des Elements auf den Clientknoten B, C trifft. Dort werden die beiden Clients ebenfalls auf den Stack gelegt und ihre *visited* Flage gesetzt (Situation b). Der gegenüber o_B äußere Bereich von E_1 erreicht nochmals einen Clientknoten mit C. Da dessen Flag aber bereits gesetzt ist, kommt es nicht zu einem Eintrag auf dem Stack. Somit ist der Test von E_1 beendet und der Stack enthält genau die drei Clients A, B und C, an die E_1 zu versenden ist.

Ehe nun E_2 gegen den durch den *Split Node* markierten Teilbaum getestet wird, ist der Stack auf den alten Zustand zu restaurieren. Daher erfolgt eine Umsetzung des Stack Pointers auf die Position, als der *Split Node* erreicht wurde (Situation c). Zuvor sind jedoch alle *visited* Flags der Clients zwischen der aktuellen und der alten Position des Stack Pointers wieder auf *False* zu setzen. Anschließend kann der Test von E_2 beginnen. E_2 liegt außerhalb von B aber im inneren von C. Sobald E_2 den Clientknoten von C erreicht, wird C auf dem Stack eingetragen (Situation d).

Der *Client Stack* kann niemals die Anzahl der registrierten Clients übersteigen, weshalb eine Preallokation des Stacks möglich ist. Dadurch werden aufwändige Speicheroperationen vermieden.

7.6.1 Implementierung des Out-of-Core Konzepts

Innerhalb des Systems gibt es keine zentrale Komponente, die das Aus- und Einlagern der Daten kontrolliert. Vielmehr ist die Out-of-Core Strategie implizit gegeben, da sämtliche speicherintensiven Datenstrukturen wie der Szenegraph und die Pools [7] mittels der in Abschnitt 7.2.2 beschriebenen Key-Value Maps verwaltet werden. Der Scheduler wertet nun die Areas of Interest aller registrierten Clients aus, wobei die Areas of Interest genau die relevanten Bereiche der Szene darstellen, welche sich möglichst im Hauptspeicher befinden sollten. Da der Scheduler beim Auswerten exakt auf diese Regionen des Szenegraphen zugreift und die Maps eine Verdrängungsstrategie fahren, befinden sich die benötigten Informationen in der Regel automatisch im Hauptspeicher. Voraussetzung ist hierfür allerdings eine möglichst stetige Navigation der Benutzer.

[7] Elementgraphen werden auch als Pooleinträge verwaltet.

7.7 Implementierung von Multiplexer und Demultiplexer

Wie schon in Abbildung 7.20 angedeutet übergibt der Scheduler dem Multiplexer in jedem Zyklus diejenigen *LeafNodes*, welche sich innerhalb der Area of Interest des Betrachters befinden. Weitere Informationen sind die ID des Clients, an den der *LeafNode* zu versenden ist, die Priorität des *LeafNodes* sowie der Zyklus. Der Scheduler wirkt also wie eine Art Trigger, unabhängig ob der *LeafNode* bereits im Multiplexer vermerkt ist oder nicht. Die Ursache hierfür liegt in dem Problem, dass sich die Priorität eines *LeafNodes* auch noch während seiner Übertragung verändern kann, d.h. obwohl er schon im Multiplexer registriert ist. Der Multiplexer muss also permanent über variierende Prioritäten der *LeafNodes* auf dem Laufenden bleiben.

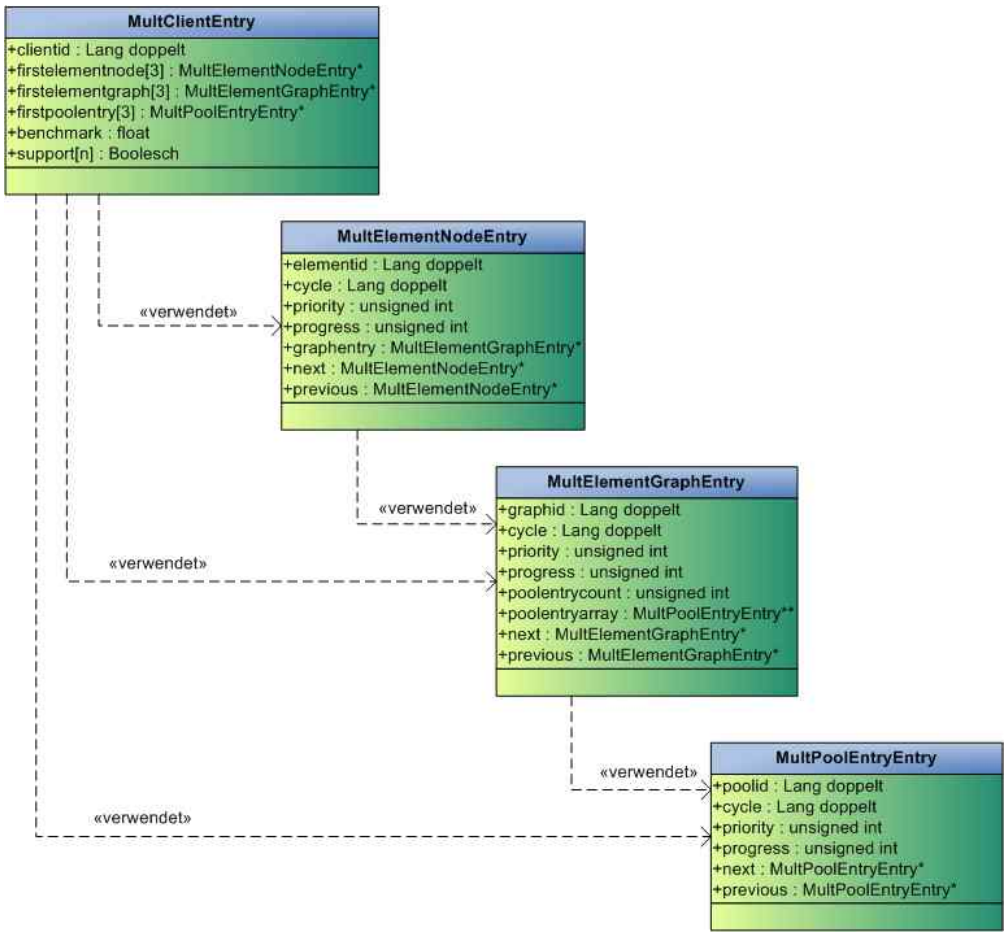

Abbildung 7.22: *Die vereinfachte Klassenstruktur der Einträge innerhalb des Multiplexers. Auf der Seite des Demultiplexers existieren ähnliche Gegenparts.*

Mit einem *LeafNode* sind in der Regel zusätzliche Informationen verbunden. Handelt es sich im Falle des *LeafNodes* um einen *ElementNode*, so verweist letzterer auf einen Elementgraphen und dieser wiederum auf mehrere Pooleinträge. *GroupNodes* erfordern eine weitere Stufe, da sie eine Hierarchie an *ElementNodes* adressieren. Dieser Abschnitt geht der Einfachheit halber im Wesentlichen auf die Behandlung der *ElementNodes*, da das vorgestellte Prinzip sich einfach auf die *GroupNodes* erweitern lässt. Für die Übertragung eines *Element-*

Nodes an einen Client erstellt der Multiplexer mittels der in Abbildung 7.22 dargestellten Datenstrukturen einen Abhängigkeitsgraphen. Der *MultClientEntry* wird einmal zu Beginn der Registrierung eines Clients erzeugt und beinhaltet die für den Scheduler relevanten Informationen des Clients, beispielsweise die ID des Clients. Der *MultClientEntry* verfügt über neun Listen, wobei jeweils drei den *ElementNodes*, drei den Elementgraphen und drei den Pooleinträgen vorbehalten sind. Jede der drei Listen eines Typs repräsentiert entsprechend den drei Zonen einer Area of Interest die Priorität ihrer Elemente. Die Verkettung der Einträge erfolgt analgo zu den Knoten des Szenegraphen, d.h. die im Folgenden beschriebenen Strukturen können sich maximal in einer der Liste befinden und beinhalten dafür ein *previous* beziehungsweise ein *next* Feld. Der *MultElementNodeEntry* verwaltet die Informationen eines *ElementNodes* und adressiert einen *MultElementGraphEntry*, welcher die Daten eines Elementgraphen aufnimmt. Hier wiederum steht ein Array von Verweisen auf *MultPoolEntrys* mit den Daten der Pooleinträge. Elementgraphen und Pooleinträge übernehmen die Priorität der *ElementNodes*. Ein Abhängigkeitsgraph besteht immer aus mindestens einem *MultElementNodeEntry*, der einen *MultElementGraphEntry* referenziert und dieser auf mindestens einen *MultPoolEntry* verweist [8]. Die Bezeichnung Abhängigkeitsgraph ist durch die Beziehungen der Einträge hinsichtlich ihrer Übertragung gegeben. So ist ein *ElementNode* erst dann vollständig übertragen, wenn zuvor sein Elementgraph und davor wiederum die betroffenen Pooleinträge vollständig versendet wurden.

Da mehrere *ElementNodes* sich den gleichen Elementgraphen beziehungsweise mehrere Elementgraphen die gleichen Pooleinträge teilen können, ergibt sich die Priorität eines Elementgraphen durch das Maximum der Prioritäten aller ihn referenzierenden *ElementNodes*. Analoges gilt hinsichtlich der Elementgraphen und der Pooleinträge. Das Data Sharing der Einträge ist auch der Grund, weshalb der *MultClientEntry* über direkte Verweise auf Einträge aller drei Typen verfügt. Prinzipiell würde es ja reichen, im *MultClientEntry* lediglich auf die *MultElementNodeEntrys* zu verweisen und von dort die Abhängigkeitsgraphen auszuwerten. Das Auswerten der Beziehungen zwischen den einzelnen Abhängigkeitsgraphen wäre bei dieser Vorgehensweise aber zu aufwändig. Das Erkennen gemeinsamer Daten bietet den Vorteil, beispielsweise einen Elementgraphen nur einmal an einen Client übertragen zu müssen, obgleich mehrere *ElementNodes* ihn referenzieren. Der *MultClientEntry* dient daher als Sammelbecken aller an einen Client zu übertragenden Informationen und spiegelt zudem deren Priorität wider.

Jedesmal wenn der Scheduler einen *LeafNode* an den Multiplexer übergibt, testet der Multiplexer, ob ein entsprechender Eintrag bereits existiert. Bei Nichtvorhandensein wird ein neuer Eintrag erzeugt. Anschließend überprüft der Multiplexer sowohl für den LeafNode als auch für die Einträge des korrespondierenden Abhängigkeitsgraphen, ob die mit dem *LeafNode* angegebene Priorität größer als die aktuell vermerkte Priorität ist. Ist die neue Priorität größer, dann wird der alte Wert dem neuen Wert angepasst. Zudem wird das Attribut *cycle* in den betroffenen Einträgen auf den momentanen Zyklus gesetzt, welchen der Scheduler zusammen mit dem *LeafNode* dem Multiplexer übergibt. Erhält ein Eintrag des Abhängigkeitsgraphen eine höhere Priorität, so erfolgt zudem ein Wechsel des Eintrages in eine Liste

[8] Prinzipiell ist natürlich auch ein Elementgraph ohne Pooleinträge möglich, jedoch hätte dieser kein äußeres Erscheinungsbild. Elemente mit derartigen Elementgraphen können beispielsweise in Animationshierarchien als Dummy Körper auftreten

des *MultClientEntry* mit entsprechender Priorität. Entspricht die neue Priorität dem alten Wert, so wird lediglich der Zyklus aktualisiert. Fällt die neue Priorität dagegen geringer als die alte aus, dann bleibt alles unverändert.

Aus dieser Vorgehensweise ergibt sich die Frage, wie denn nun die Priorität eines Eintrages wieder sinken kann. Da mehrere *ElementNodes* den gleichen Elementgraphen adressieren können, wäre es fatal, die Priorität eines Elementgraphen sofort zurückzustufen, nur weil die Priorität eines der *ElementNodes* momentan niedriger ausfällt. Dies könnte zu Flattereffekten führen, bei denen die Priorität der Einträge ständig variiert. Stattdessen überprüft der Multiplexer über die Differenz aus dem aktuellem Zyklus des Schedulers und dem *cycle* Feld des Eintrages, wie lange das *cycle* Feld nicht mehr aktualisiert wurde. Dies ist gleichbedeutend mit einer Überprüfung, wie lange die im Eintrag vermerkte Priorität nicht mehr erreicht wurde. Übersteigt die Differenz der Zyklen einen Schwellwert, so wird der betroffene Eintrag um eine Prioritätsstufe zurückgesetzt.

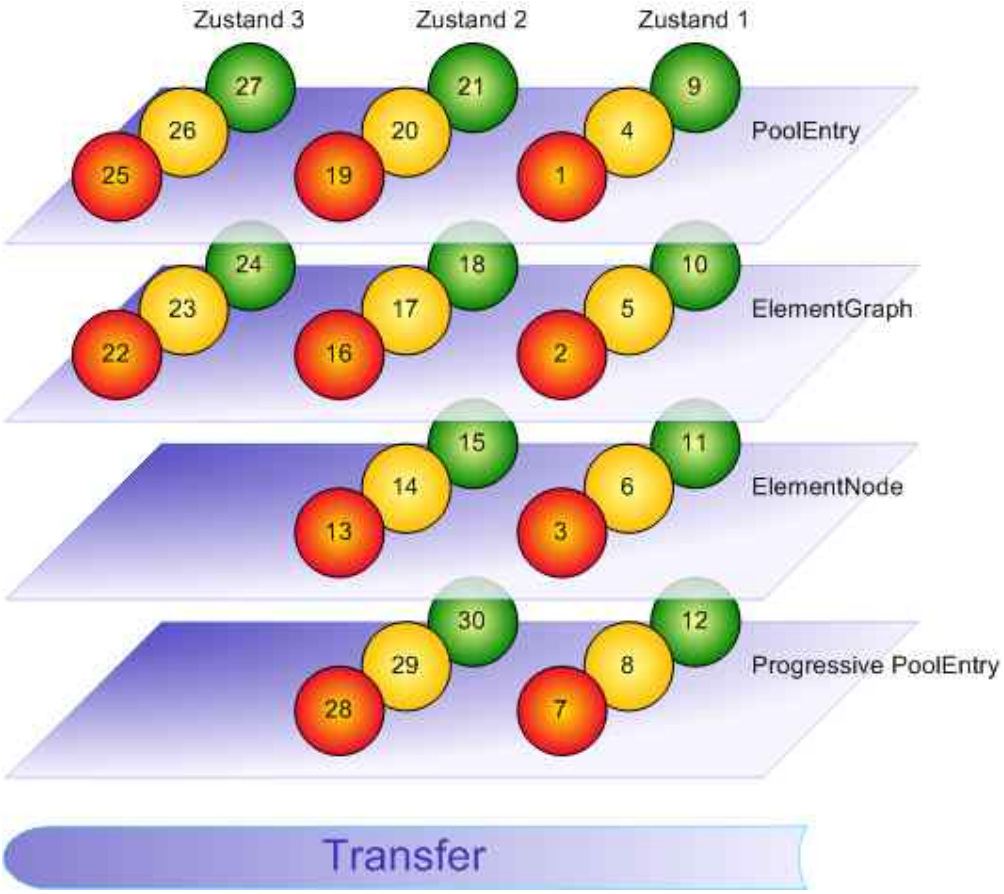

Abbildung 7.23: *Ein wichtiges Kriterium für eine effiziente Übertragung ist die Reihenfolge, in welche die einzelnen Protokolle der Daten durch den Server gefahren werden. Die Nummern auf den einzelnen Bausteinen geben die Bedeutung hinsichtlich der Übertragung wieder. Die Farben symbolisieren die wichtigste Zone der möglicherweise mehreren Areas of Interest, in denen sich die Bausteine befinden.*

Neben der Priorität eines Eintrages ist auch seine Position innerhalb des Abhängigkeitsgraphen für die Übertragung von Bedeutung. Die Reihenfolge, in welcher die einzelnen Einträge vom Multiplexer bearbeitet werden, ist in Abbildung 7.23 illustriert. Die vertikale Richtung

der Abbildung stellt den Abhängigkeitsgraphen eines *ElementNodes* da, d.h. der Multiplexer beinhaltet einen *MultElementNodeEntry*, einen *MultElementGraphEntry* und einen bis mehrere *MultPoolEntrys*. Möglicherweise verweist der Elementgraph auf progressive Pooleinträge, welche nochmals einer gesonderten Behandlung bedürfen. Die Reihenfolge von oben nach unten entspricht der Position im Abhängigkeitsgraph. Als erstes sind somit die Pooleinträge zu übertragen. Zum einen können Pooleinträge von mehreren Elementgrapgen referenziert werden und daher von für eine Vielzahl von *ElementNodes* von Bedeutung sein. Zum anderen ist ein Elementgraph erst dann visualisierbar, wenn alle seine Pooleinträge vollständig übertragen sind. Gleiches gilt für die Bezeihung zwischen *ElementNode* und Elementgraph. Eine Ausnahme bilden progressive Pooleinträge. Hier ist zunächst nur der Transfer der Basisinformation wichtig, welche für eine - wenn auch grobe - Visualisierung ausreicht. Im Bezug auf progressive Pooleinträge sind also zunächst die Basisinformationen, der Elementgraph und der *ElementNode* zu versenden, weil bereits dann eine Visualisierung möglich ist. Die Übertragung weiterer progressiver Daten erfolgt erst im Anschluss. Aus diesem Grund befinden sich die progressiven Pooleinträge in der untersten Position des Abhängigkeitsgraphen. Ein wichtigen Einfluss auf die Reihenfolge der Übertragung hat die Priorität eines Eintrages. Die Priorität wird entsprechend der wichtigsten Zone aller Areas of Interest gewählt, in der sich ein Eintrag befindet. Die Priorität ist in Abbildung 7.23 durch die Farben symbolisiert. Demnach wird also ein Pooleinträg innerhalb einer sichtbaren Zone als erstes übertragen. Da Elemente in einer sichtbaren Zone besonders wichtig sind, werden entsprechende Elementgraphen und *ElementNodes* gegenüber Pooleinträgen der dynamischen Zone bevorzugt.

Die horizontale Richtung in Abbildung 7.23 beschreibt die möglichen Zustände eines Eintrages während der Übertragung, welche über das *progress* Feld kodiert sind. Ein Pooleintrag verfügt also ebenso wie der Elementgraph über drei Zustände, wohingegen *ElementNodes* und progressive Pooleinträge lediglich zwei Transferzustände aufweisen. Die Gliederung der Zustände ist für eine saubere Kommunikation zwischen Client und Server erforderlich und spiegelt das Übertragungsprotokoll für einen Eintrag wieder. Der erste Zustand eines Eintrages bedeutet, dass mit der Übertragung des Eintrages noch nicht begonnen wurde. Den zweiten Zustand erreicht ein Eintrag im Multiplexer, sobald selbiger den Transfer abgeschlossen hat und nun auf die Antwort des Clients wartet, welche den vollständigen Empfang der Daten verifiziert. Dieser Zustand fehlt bei den progressiven Pooleinträgen, da hier noch sehr viel später weitere Daten folgen können. Der dritte Zustand eines Eintrages weist den Multiplexer darauf hin, dem Demultiplexer des Clients eine Nachricht zukommen zu lassen, wonach die Kommunikation über den Eintrag nun beendet ist und der Demultiplexer seinen internen Strukturen hinsichtlich des Eintrages löschen kann.

Abbildung 7.24 illustriert das Übertragungsprotokoll am Beispiel eines Pooleintrages. Wie in Abschnitt 7.3.2 beschrieben, verfügen zwei korrespondierende Einträge auf Client und Server wegen der Verwendung der Key-Value Maps über unterschiedliche Identifikationen. Der Multiplexer sendet einen Pooleintrag einschließlich seiner serverseitigen ID an den Client und versetzt den Pooleintrag dadurch in den zweiten Zustand. Der Demultiplexer des Clients verwaltet den Pooleintrag unter der ID des Servers und wartet auf die Vollständigkeit der Daten. Danach trägt er den Pooleintrag in den Pool des Clients ein und erhält so eine clientseitige ID. Diese ID sendet der Client an den Server als Verifikation für die Richtigkeit der Daten. Die ID wird im Pooleintrag vermerkt und symbolisiert dort die vollständige

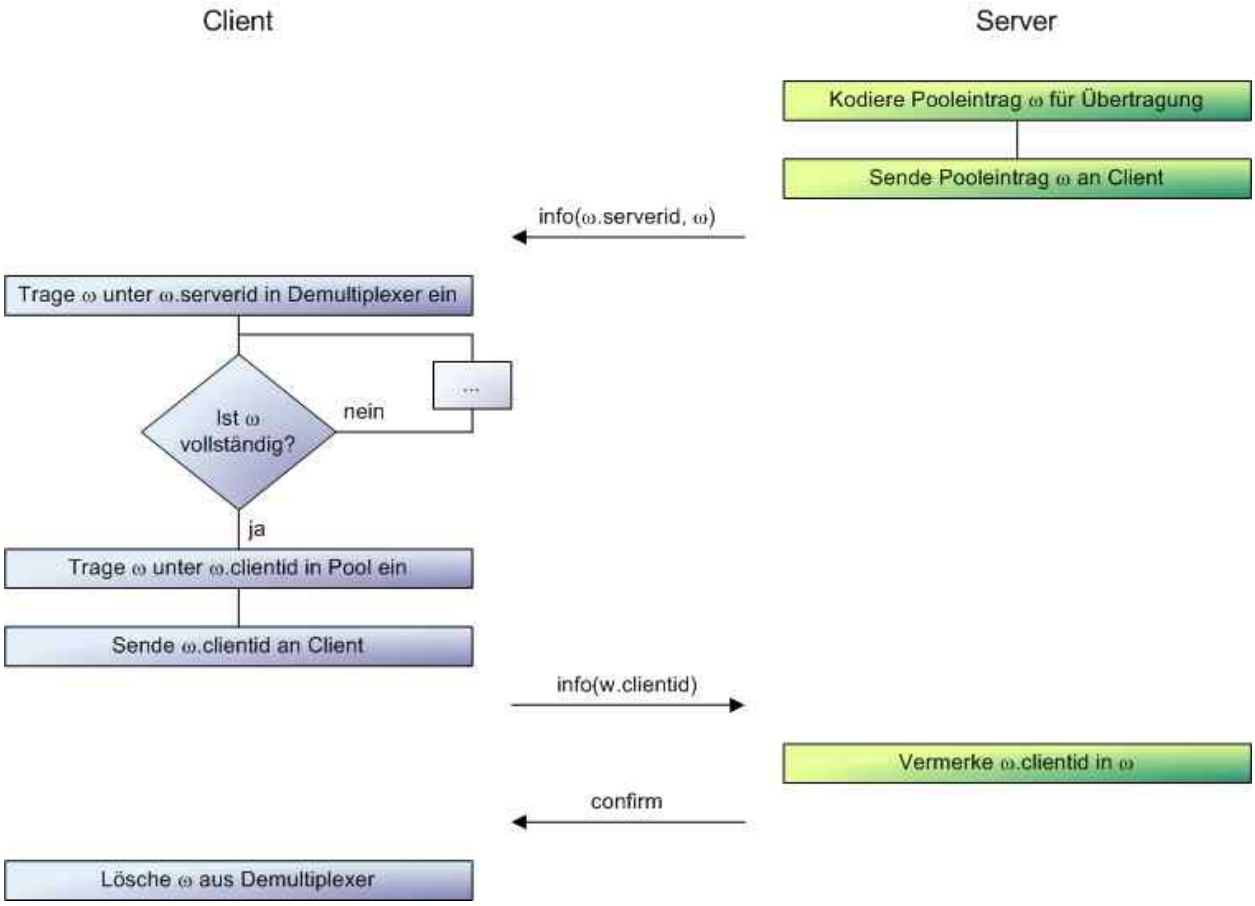

Abbildung 7.24: *Das Übertragungsprotokoll der Pooleinträge ist analog zu den Elementgraphen und den ElementNodes.*

Übertragung des Eintrages an den Client. Nach Erhalt der Antwort befördert der Multiplexer den Eintrag in den dritten Zustand, welcher eine Bestätigung an den Client zur Folge hat. Der Client löscht nun den Eintrag aus dem Demultiplexer. Sollte im Folgenden nun eine weitere Kommunikation über diesen Pooleintrag zwischen Client und Server erforderlich sein, so erfolgt selbige mittels der clientseitigen ID, welche beiden Parteien bekannt ist.

Vor dem Transfer eines Elementgraphen traversiert der Multiplexer mittels eines Renderers den Graphen und überprüft, welche Knoten von dem betroffenen Client unterstützt werden. Nicht unterstützte Knoten wie etwa Textur- oder Shaderknoten werden durch den Renderer entfernt, womit die referenzierten Pooleinträge ebenfalls nicht zur Übertragung gelangen. Der Multiplexer speichert im *support* Feld des *MultClientEntry* eine Liste mit Flags, welche die möglichen Knoten identifizieren. Die zu übertragende Datenmenge progressiver Pooleinträge wird mittels des *benchmark* Attributes berechnet. Dabei handelt es sich um einen prozentualen Faktor, welcher die Leistungsmerkmale des Endgerätes und der Netzverbindung berücksichtigt. Aufgrund der Dynamik der Bandbreite wird dieser Wert von Zeit zu Zeit aktualisiert. Bevor der Multiplexer einen Pooleintrag versendet, wählt er für den Eintrag einen geeigneten Codec aus [9]. Der Codec kodiert und komprimiert die Informationen, worauf

[9]Der Designer eines Modelles sollte für jeden benötigten Pooleintrag einen bis mehrere potentielle Codec

der Multiplexer die Daten an die Netzwerkkomponente weiterleitet. Aufgrund der möglichen Wartezeiten beim Versenden und Empfangen von Nachrichten arbeitet der Multiplexer mit einem Threadpool.

Die Implementierung des Demultiplexers auf Seite des Clients ist nahezu identisch mit der des Servers. Allerdings dienen die Codecs hier der Dekodierung der Informationen.

7.8 Implementierung der I/O Plugins und Codecs

Die Realisierung der Plugins und Codecs ist nahezu identisch: Beide Komponententypen sind als dynamische ACE Dynamic Link Library implementiert, welche zur Laufzeit geladen und wieder gelöscht werden können. Hierdurch ist eine dynamische Erweiterung der Systemfunktionalität möglich. Während die I/O Plugins für das Laden und Speichern bestimmter Dateiformate verantwortlich sind, dienen die Codecs zur Kodierung und Dekodierung der Daten vor beziehungsweise nach deren Übertragung. Mit jedem I/O Plugin oder Codec ist genau ein Format verbunden. Der jeweilige Algorithmus wird dabei durch die Komponente gekapselt.

Sowohl für die I/O Plugins als auch die Codecs existieren auf Seite von Client und Server entsprechende Manager, welche die Codecs verwalten und zur Verfügung stellen. Diese Manager bestehen im Wesentlichen aus einer Key-Value Map. Unter Angabe der Bezeichnung eines Dateiformats oder eines Codecnamens gibt der jeweilige Manager einen Verweis auf die gewünschte Komponente frei. Eine Liste dieser Codecnamen ist ebenfalls in den Pooleinträgen gespeichert, um eine geeignete Kodierung vor der Übertragung im Multiplexer auszuwählen.

Die Schnittstelle der Codecs ist sehr allgemein gehalten. Sie besteht aus zwei Methoden *encode* und *decode*, wobei erstere die Daten kodiert und letztere dekodiert. Hierzu erhalten beide Methoden einen *void* Pointer, welcher die Quelldaten referenziert. Weiterhin wird die Länge der Quelldaten übergeben. Beide Methoden liefern einen Pointer auf die resultierenden Informationen sowie deren Länge zurück.

Die Schnittstelle der I/O Plugins fällt analog aus, allerdings sind die beiden Methoden hier als *open* und *save* bezeichnet.

7.9 Implementierung der Netzwerkanbindung

Die Implementierung der Netzwerk-Komponenten von Client und Server basiert auf einer Task-orientierten Vorgehensweise. Die übrigen Komponenten erstellen einen Task beziehungsweise eine Aufgabe und erteilen selbige dann der Netzwerk-Komponente. Diese sammelt

auswählen. Ansonsten wird je nach Datentyp automatisch ein Standardcodec ausgewählt. Jeder Pooleintrag speichert die Identifikationen der Codecs.

die Tasks in einer Bearbeitungsschlange und bearbeitet die Aufgaben nach dem First-In-First-Out (FIFO) Prinzip. Die Taskqueue der Netzwerk-Komponente wird von einem einzelnen Thread überwacht, welcher anfallende Tasks an sogenannte Netzwerk-Agenten weiterleitet. Jede Netzwerk-Komponente verfügt hierzu über einen Pool an Netzwerk-Agenten. Ein Netzwerk-Agent verfügt über einen eigenen Thread und begleitet selbstständig einen Task bis zu dessen Ende. Hierunter ist ein erfolgreicher Abschluss der Aufgabe, ein Fehler während der Bearbeitung oder das Abbrechen durch den Benutzer zu verstehen. Mögliche Tasks sind beispielsweise das Versenden von Nachrichten oder das Erstellen einer Verbindung zwischen Client und Server. Im Falle eines Fehlers beim Versenden einer Nachricht versucht der Netzwerk-Agent den Fahler abzufangen und die Nachricht erneut zu versenden. Für jeden möglichen Fehler ist ein Timeout definiert, welches beim Erreichen das Scheitern der Aufgabe signalisiert. Tasks können durchaus komplexer sein und etwa das mehrmalige Versenden von Nachrichten erfordern. Grundlegendes Ziel eines Tasks ist die Beschreibung einer vollständigen Aufgabe, die durch den Agenten zu lösen ist.

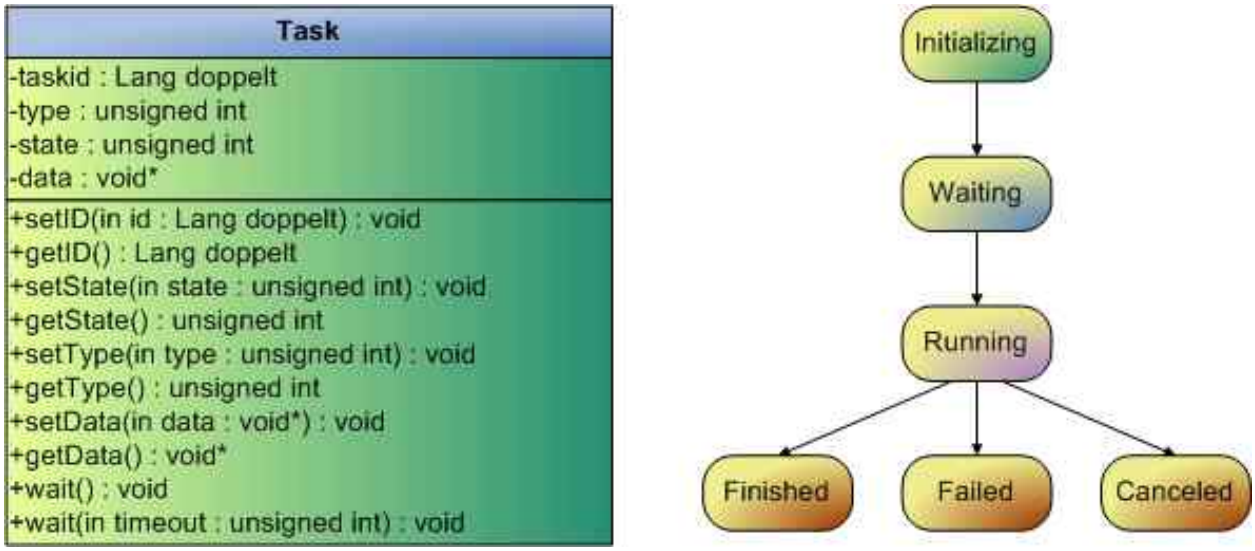

Abbildung 7.25: *Das Klassen- und Zustandsdiagramm der Tasks innerhalb der Netzwerk-Komponente.*

Abbildung 7.25 illustriert die Struktur der *Task* Klasse sowie das Zustandsdiagramm eines Tasks. Jeder Task verfügt über eine eindeutige ID, einen Typ, einen Zustand sowie aufgabenspezifische Daten. Der Typ definiert die Art der Aufgabe, beispielsweise das Versenden einer Nachricht. Zu Beginn befindet sich ein Task immer im Zustand der Initialisierungsphase, d.h. er ist noch nicht der Netzwerk-Komponente zugeteilt. Sobald sich ein Task in der Warteschlange der Netzwerk-Komponente befindet, nimmt er den Zustand *Waiting* an. Sobald die Bearbeitung des Tasks beginnt, befindet der Task sich im Zustand *Running*. Anschließend wird der Task mit einer der drei bereits erwähnten Zustände beendet.

Tasks bieten den anderen Komponenten, d.h. den Auftraggebern, die Möglichkeit, auf die Ergebnisse der Aufgabe zu warten. Dies erfolgt über den Aufruf der *wait* Methode. Um aktives Polling zu vermeiden ist hier ein Semaphore Pattern implementiert. Der beauftragende Thread wird an einem Semaphore blockiert, welcher durch den bearbeitenden Thread des Netzwerk-Agenten wieder freigegeben wird. Die *wait* Methode erlaubt somit die Synchroni-

sation zwischen Client und Server. Im Falle des Versenden von Nachrichten können die Auftraggeber unter der ID des Tasks einen Briefkasten bei der jeweiligen Netzwerk-Komponente anlegen und sich dort die eintreffende Post abholen. Die Netzwerk-Komponente legt die empfangenen Nachrichten automatisch in dem richtigen Postfach ab.

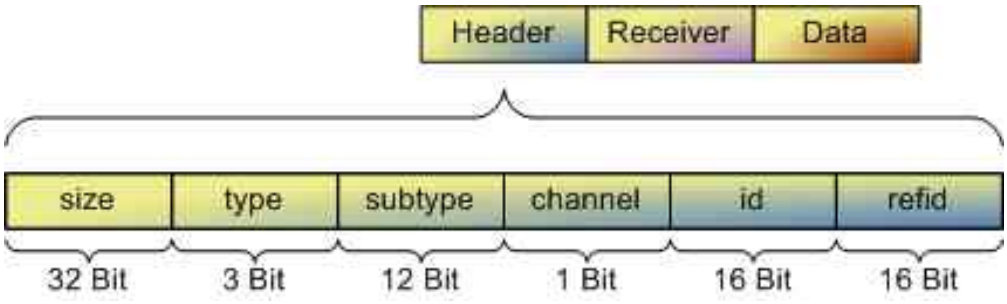

Abbildung 7.26: *Die Struktur der Nachrichten.*

Abbildung 7.26 stellt die Struktur der Nachrichten dar, welche in einen Header und einen Datenteil gegliedert ist. Der *Receiver* beinhaltet die ID des Zieles und wird nicht versendet, da diese Information implizit durch die Repräsentation der Verbindung zwischen Client und Server gegeben ist. Gleiches gilt für die Identifikation des Senders der Nachricht. Der Header verfügt über eine konstante Länge und beinhaltet wichtige Informationen für das Versenden und Verwalten der Nachrichten. Das Attribut *size* beschreibt die Länge der auf einen Header folgenden Daten. Diese Information ist für den Empfänger von Bedeutung, um das Ende der eintreffenden Nachricht zu identifizieren. Der *type* beschreibt die Art der Nachricht, wobei es sich um einen Request, ein Inform, ein Accept, ein Reject oder ein Reply handeln kann. Das Attribut *subtype* kodiert zusätzlich einen speziellen Untertyp innerhalb des gewählten Haupttypes. Ob die Nachricht über den Daten- oder Kontrollkanal zu versenden ist, wird mittels des *channels* selektiert. Aus Sicht der Netzwerk-Komponente ist es transparent, ob der angegebene Datenteil einen Befehl oder zum Beispiel geometrische Informationen beherbergt. Die *id* identifiziert eine Nachricht und ist wichtig für komplexere Kommunikationen. In diesem Fall ist es erforderlich auf eine vorangehende Nachricht Bezug zu nehmen. So bezieht sich beispielsweise ein Reply immer auf einen Request. Um ein derartiges Verhältnis darzustellen, beinhaltet der Header eine *refid*, welche die ID der verursachenden Nachricht beinhaltet.

Die Aufgabe zur Erstellung einer Verbindung zwischen Client und Server wird durch den Benutzer auf Seite des Clients erteilt. Der durch die Netzwerk-Komponente des Clients beauftragte Netzwerk-Agent versucht zwei Kanäle zwischen Client und Server zu etablieren, einen Kontroll- und einen Datenkanal. Die Kanäle werden im System durch eigene Klassen repräsentiert und kapseln das zugrunde liegende Protokoll der Verbindung. Sie kümmern sich beispielsweise um die richtige Reihenfolge eintreffender Pakete, mögliche Ausfälle einzelner Pakete und über das Benachrichtigen der Netzwerk-Komponente über eintreffende Informationen. Durch das *Reactor* Konzept von ACE ist es möglich, aktives Polling zu vermeiden. Dies gilt sowohl für das Eintreffen von Informationen auf bereits etablierten Verbindungen als auch für das Gesuch, eine neue Verbindung aufzubauen. Letzteres ist mittels der ACE *Aceptor* Struktur realisiert. Die Kanäle selbst sind über Socket Verbindungen implementiert. Das *Reactor* Konzept von ACE gleicht der ereignisorientierten Vorgehensweise, d.h. sobald

auf einem Kanal eine Information auftritt, wird intern ein Ereignis ausgelöst und eine vom Programmierer angelegte Methode aufgerufen. Die Methode beinhaltet die Reaktion auf das Ereignis.

7.10 Implementierung der I/O-Komponente

Die Implementierung der I/O-Komponente fällt ähnlich der Netzwerk-Komponente aus. Auch hier existiert eine Taskqueue, die von einem einzelnen Thread bearbeitet wird. Die einzelnen Komponente vergeben an die I/O-Komponente Tasks, um Daten auf die Festplatte oder ein anderes lokales Dateisystem zu schreiben oder um Informationen von einem Medium zu lesen. Im Unterschied zur Netzwerk-Komponente allerdings existieren keine Agenten, welche die einzelnen Aufgaben in einem eigenen Kontrollfluss bearbeiten. Stattdessen werden alle Tasks von einem weiteren Thread übernommen. Somit verfügt die I/O Komponente über zwei Threads, einmal dem Thread der Taskqueue und dem Bearbeitungsthread. Durch ersteren bleibt die I/O-Komponente immer ansprechbar und kann Tasks ohne Blockade der Auftraggeber entgegennehmen. Der Bearbeitungsthread sorgt für die serielle Erledigung der Tasks. Hierdurch wird im Falle einer Festplatte ein mögliches Flattern der Köpfe verhindert, wie es etwa beim parallelen Bearbeiten der Tasks mittels mehrerer Threads geschehen kann. Analog zur Netzwerk-Komponente besteht für die Auftraggeber die Möglichkeit synchronisierender Aufrufe, d.h. die Komponente können auf das Resultat eines Tasks beziehungsweise einer Lese- oder Schreibeoperation warten.

7.11 Implementierung der Präsentation-Komponente

Dieser Abschnitt beschreibt die Implementierung der Area-of-Interest Kommunikation zwischen Client und Server sowie die Realisierung des Occlusion Cullings. Auf die Umsetzung der Navigationsschnittstelle wird nicht weiter eingegangen, da selbige mittels der QT Oberflächenbibliothek einfach zu bewerkstelligen ist.

7.11.1 Implementierung der Area-of-Interest Adaption

Die Adaption der Area-of-Interest erfolgt in einer eigenständigen Komponente, welche an die Präsentation-Komponente des Clients angebunden ist. Diese Komponente erfasst in regelmäßigen Abständen die Position und die Blickrichtung des Betrachters sowie die Ausmaße der Sichtpyramide. Die ermittelten Werte werden in einer Tabelle vermerkt, wobei der älteste Wert durch die aktuellen Ergebnisse verdrängt wird. Anhand dieser Historie erfolgt die Identifikation der Bewegungsart. Für jede Bewegungsart sind hierzu Toleranzen definiert, da beispielsweise ein Geradeauslaufen ohne die geringste Abweichung nur sehr selten vorkommt. Kurvenläufe werden durch das Skalarprodukt zweier Bewegungevektoren erkannt, d.h. der Winkel zwischen den Vektoren ist entscheidend.

7.11.2 Implementierung des Occlusion Culling

Das in Abschnitt 6.7.2 eingeführte Occlusion Culling gliedert sich pro Frame in drei Phasen, nämlich in das View Frustum Culling, die Selektion der Occluder sowie dem eigentlichen Occlusion Culling. Alle drei Phasen sind über Renderer implementiert, welche den Raumunterteilungsbaum traversieren. Für das View Frustum Culling und die Selektion der Occluder greifen die Renderer auf den OpenGL Feedback Modus zurück.

Sobald der Feedback Modus von OpenGL aktiviert ist, werden die über die Grafikpipeline ausgegebenen Primitive nicht mehr in den Framebuffer der Grafikkarte gerendert, sondern in Form von Werten im Feedback Buffer abgelegt. Die Primitive wurden zuvor an der Sichtpyramide geclippt, weshalb ein unsichtbares Primitiv keine Einträge im Feedback Buffer hinterlässt. Diese Eigenschaft kann über die Anzahl der Einträge abgefragt und somit als einfacher Sichtbarkeitstest im View Frustum Culling eingesetzt werden. Im Falle eines sichtbaren Primitives enthält der Feedback Buffer die Scheitelpunkte des nach dem Clipping resultierenden Primitives, wobei die x- und y-Koordinaten den Pixeln des Framebuffers entsprechen und die z-Koordinate dem normierten Distanzwert zwischen 0 und 1. Mittels der x- und y-Koordinaten ist eine sehr genaue und schnelle Berechnung der von einem Primitiv belegten Pixelanzahl möglich. Dies wird für die Selektion der Occluder ausgenutzt, da nur Primitive ab einer bestimmten Größe als Occluder geeignet sind.

Die effiziente Berechnung des Zustands *Intern* während des View Frustum Cullings erfordert eine besondere Vorgehensweise. Daher beschreibt der folgende Algorithmus die Überprüfung einer achsenparallelen, quaderförmigen Bounding Box auf ihren Sichtbarkeitszustand:

1. Alle Scheitelpunkte der Bounding Box werden im Feedback Modus gerendert. Liefern alle Scheitelpunkte im Feedback Buffer Ergebnisse, dann ist die Bounding Box *Vollständig Sichtbar* und der Algorithmus wird beendet.

2. Eine neue Projektion mit einer orthogonalen 3D Projektion wird definiert. Die Sichtpyramide ist durch die Bounding Box selbst repräsentiert mit Blickrichtung entlang der negativen z-Achse und einem Aufwärtsvektor entlang der positiven y-Achse. Anschließend wird die Projektionsfläche des Betrachters im Feedback Modus gerendert. Kommt es zu einem Eintrag im Feedback Buffer, dann ist die Bounding Box *Intern* und der Algorithmus wird beendet.

3. Die alte Projektionsmatrix wird wieder eingesetzt. Die dem Betrachter zugewandten Oberflächen der Bounding Box werden wie in Abschnitt 6.7.2 bestimmt und der Reihe nach im Feedback Modus gerendert. Sobald ein Eintrag im Feedback Buffer existiert, ist die Bounding Box *Partiell Sichtbar* und der Algorithmus wird beendet.

4. Die Bounding Box erhält den Zustand *Unsichtbar*.

Nach dem View Frustum Culling werden die zumindest teilweise sichtbaren Bereiche der Szene auf Occluder überprüft. Unglücklicherweise geben die meisten Grafikchips ein im Feedback Modus gerendertes Primitiv in Form mehrerer Fragmente zurück. Die Ursache hierfür liegt in der auf Dreiecke beschränkten Verarbeitung der Grafikchips. Üblicherweise werden

die Primitive nämlich entweder noch vor oder spätestens nach dem Clipping an der Sichtpyramide in Dreiecke tesseliert. Ein als Occluder erkanntes Primitiv erfordert daher eine Rekonstruktion anhand der einzelnen Fragmente. Abbildung 7.27 zeigt zwei mögliche Unterteilungen der Primitive in Fragmente. Während der linke Fall eine reine Gliederung in Dreiecke aufweist, erzeugen andere Grafikkarten wie etwa die Wildcat durchaus auch Fragmente mit mehr als drei Scheitelpunkten. Hierdurch ist eine einfache Rekonstruktion wie im Fall I. nicht mehr möglich. Stattdessen müssen alle doppelten Kanten und Scheitelpunkte der einzelnen Fragmente entfernt und im Anschluss die Fragmente wieder in der richtigen Orientierung zusammengefügt werden. Dieser Prozess erfordert mehr Aufwand, da aber eine Rekonstruktion aufgrund der beschränkten Zahl der Occluder selten benötigt wird, sind die Auswirkungen auf die Laufzeit gering.

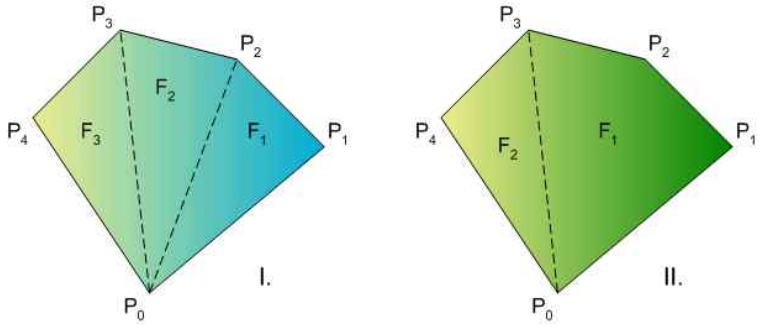

Abbildung 7.27: *Je nach Grafikchip kann die Tesselierung eines Primitives unterschiedlich ausfallen.*

Vor dem Einfügen der selektierten Occluder in den Occlusino Tree erfolgt eine Sortierung der Occluder mittels eines BSP Verfahrens in eine Front-To-Back Reihenfolge sortiert, da sich die Polygone der Szene gegenseitig durchdringen können. Das Clipping sowohl im BSP Tree als auch später im Occlusion Tree geschieht nach einem erweiterten Sutherland-Hodgman Algorithmus. Der Algorithmus erkennt nicht nur, ob ein Polygon sich vollständig auf der inneren oder äußeren Seite einer Kante oder Ebene befindet. Das Verfahren kann ein Polygon auch dann noch einer Seite eindeutig zuordnen, wenn wie in Abbildung 7.28 eine Kante des Polygons auf der Schnittkante oder -ebene h verläuft. Somit schneidet der Algorithmus derartige Polygon nicht an h, wodurch der Clippingaufwand und damit auch die Tiefe sowohl des BSP Baumes als auch des Occlusion Trees deutlich reduziert wird.

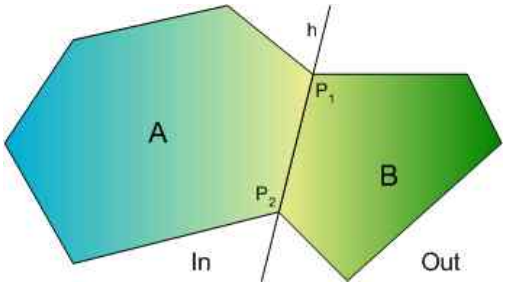

Abbildung 7.28: *Der erweiterte Sutherland-Hodgman Algorithmus klassifiziert auch dann Polygone als In oder Out, wenn eine Kante des Polygons auf der Schnittkante beziehungsweise Schnittebene h verläuft.*

7.12 Zusammenfassung

Dieses Kapitel dient zur Beschreibung der Implementierung des im vorangehenden Kapitel vorgestellten Konzepts. Hierzu wird zunächst auf einige verwendete Werkzeuge eingegangen, wobei es sich um die *Standard Template Library* (STL), *Adaptive Coomunication Environment* (ACE), Qt, OpenGL, *OpenGL Utility Toolkit* (GLUT), Freeglut sowie *DjVu* handelt. Die STL ist ein Bibliothek für wichtige Basisklassen wie Vector, Map, oder Set. Die ACE Bibliothek erlaubt die Implementierung von Netzwerk und Multithreading auf vielen wichtigen Plattformen. Qt ist eine Oberflächenbibliothek zur Definition von grafischen Benutzeroberflächen. OpenGL stellt eine Grafikpipeline zur Visualisierung von Szenen dar. Die GLUT und Freeglut Bibliotheken geben hierzu eine erweiterte Funktionalität. DjVu ist eigentlich ein Format für die progressive Kodierung ganzer Dokumente. Der entsprechenden Bibliothek wurde das progressive Texturformat entnommen und optimiert.

Innerhalb dynamischer Szenen kommt es oftmals zu einer Vielzahl an Ein- und Ausfügeoperationen. Bedingt durch die Speichergröße ist zudem das Löschen und Erzeugen von Objekten erforderlich. Der vom Betriebssystem bereitgestellte Heap ist hierfür zu langsam, weshalb eine eigene Speicherverwaltung entwickelt wurde. Das Verfahren verzichtet auf eine Defragmentierung des Speichers und weist sowohl beim Einfügen als auch beim Ausfügen eine Komplexität von $O(n)$ auf. Wichtiger Bestandteil der Speicherverwaltung ist die Key-Value Map mit konstanter Zugriffszeit. Ermöglicht wird dies durch einen Schlüssel, welcher innerhalb des Baumes der Map den Pfad von der Wurzel bis zu einem Element beschreibt. Der Map steht ein bestimmter Speicherbereich zur Verfügung, weshalb bei einem Überlauf von Informationen einige Daten automatisch ausgelagert werden. Die Map verfährt dabei nach einem Verdrängungsprinzip, wobei die am längsten unbenutzten Information die ersten Kandidaten für eine Auslagerung sind. Beim Auslagern von Daten wird versucht, möglichst Einheiten konstanter Länge in einer Datei zu speichern, um das Überschreiben von Informationen zu ermöglichen und so ein Anwachsen der Daten zu verhindern.

Nach der Speicherverwaltung folgt die Implementierung des Szenegraphen. Alle Knoten des Elementgraphen erben von einer gemeinsamen Superklasse und bieten eine Schnittstelle für Renderer. Im Gegensatz zum Elementgraphen gibt es bei den Pooleinträgen jeweils eine server- und eine clientspezifische Implementierung. Die Ursache hierfür liegt darin, dass der Server Buch führen muss, welche Einträge bereits an einen bestimmten Client übertragen wurden. Die zentrale Anlaufstelle stellt der *Poolmanager* dar, welcher sämtliche Pooleinträge mittels einer Key-Value Map verwaltet. Hieraus ergibt sich das Problem unterschiedlicher Identifikationen für korrespondierende Elemente auf Seite von Client und Server. Der Szenegraph bietet eine einfache Schnittstelle, mit der Elemente in den Szenegraphen eingefügt, aus dem Szenegraphen ausgefügt oder innerhalb des Graphen transformiert werden können. Zudem bietet der Szenegraph die Möglichkeit zur Reorganisation, um die räumliche Struktur wieder auf einen aktuellen Stand zu bringen. Auch hier erben alle Knoten von einer gemeinsamen Superklassen. Die Vererbungshierarchie wurde so angelegt, dass eine beliebige Implementierung eines Raumunterteilungsbaumes zum Einsatz kommen kann. Die Berechnung der Indizes zum Einfügen eines Elements in eine bestimmte Zelle ist mit einem effizienten Algorithmus möglich. Somit ist keine aufwändige Suche innerhalb der potentiell 27 Kinder eines inneren Knoten erforderlich. Die Reorganisation behandelt nur die von ei-

ner Veränderung betroffenen inneren Knoten des Szenegraphen. In der ersten Phase werden alle Elemente behandelt, die einen inneren Knoten verlassen. Die zweite Phase bearbeitet sämtliche Elemente, welche eine neue Zelle betreten.

Die Implementierung der Renderer ermöglicht den Austausch verschiedener Strategien zur Laufzeit. Die Vorgehensweise erlaubt die einfache objektorientierte Erweiterung und Pflege der Renderer sowie des Szenegraphen. Neue Knotentypen können somit in das bestehende System ohne Einfluss auf die existierende Software integriert werden.

Die progressive Simplifizierung führt sowohl für die Scheitelpunkte als auch für die Dreiecke zwei grundlegende Datenstrukturen. Eine der Strukturen beinhaltet alle für eine Visualisierung wichtigen Informationen als konsistenter Array ohne Datenlöcher. Dieser Array ist für eine schnelle OpenGL Ausgabe geeignet. Die zweite Datenstruktur verwaltet für jeden Scheitelpunkt beziehungsweise für jedes Dreieck einen Pointer, der auf ein Objekt mit allen wichtigen Informationen wie etwa Zusatzinformationen oder Indices verweist. Diese zweite Struktur muss nicht konsistent sein und darf somit Löcher enthalten.

Der Scheduler ist ein Renderer, welcher die räumliche Struktur des Szenegraphen traversiert und dabei für jeden registrierten Client die Area-of-Interest auswertet. Das Problem der unterschiedlichen IDs von korrespondierenden Elementen und Pooleinträgen auf Server und Client erweist sich hier sogar als hilfreich, da die IDs als Flags verwendet werden können. Sie identifizieren, ob eine Information bereits an einen bestimmten Client übertragen wurde. Das Kommunikationsprotokoll zwischen Server und Client ist entsprechend aufgebaut. Der Scheduler vermeidet die redundante Selektion eines zu übertragenden Elements mittels eines *Client Stacks*. Die ausgewählten Elemente übergibt der Scheduler einschließlich der Priorität, dem Zyklus sowie der ID des Clients an den Multiplexer.

Durch den Einsatz der Key-Value Maps mit konstanter Zugriffszeit ist in Kombination mit dem Scheduler automatisch ein Out-of-Core Konzept realisiert. Die Areas-of-Interest entsprechen genau den Bereichen der Szenen, welche der Server bearbeiten, sprich an die Clients übertragen muss. Glücklicherweise sind das auch die Regionen, welche der Scheduler traversiert. Die übrigen Gebiete werden durch das Culling des Schdulers weitestgehend vernachlässigt. Über den Zugriff auf die Regionen von Seiten des Schedulers werden die entsprechenden Elemente soweit wie möglich im Hauptspeicher der Maps gehalten. Diese Elemente sind für die Übertragung vorgesehen und werden daher vom Multiplexer bearbeitet, d.h. auch die mit den Elementen assoziierten Informationen wie etwa Elementgraph und Pooleinträge werden permanent verwendet und landen daher ebenfalls im Hauptspeicher.

Multiplexer und Demultiplexer zeichnen sich für die Kommunikation zwischen Server und Client verantwortlich, um Elemente zu übertragen. Zwischen den einzelnen Informationen der Elemente besteht dabei ein Abhängigkeitsgraph. So ist es etwa sinnlos, ein Element an einen Client zu versenden, dessen Elementgraph noch gar nicht verschickt wurde. Deswegen werden die Daten in einer bestimmten Reihenfolge übertragen, wobei zusätzlich die Priorität hinsichtlich der Area-of-Interest zu berücksichtigen ist. Erschwert wird die Bestimmung der Reihenfolge durch das Data Sharing, d.h. mehrere Elementgraphen können sich einen Pooleintrag beziehungsweise mehrere Elemente den gleichen Elementgraphen teilen. Somit haben eventuell mehrere Prioritäten Einfluss auf einen Eintrag im Multiplexer. Aus diesem Grund sinkt die Priorität eines Eintrages, sofern er für eine bestimmte Anzahl von Zyklen nicht

mindestens die aktuelle Priorität erreicht. Hierdurch wird auch ein Flattern der Prioritäten vermieden. Der Multiplexer entscheidet auch, wie viele und welche Informationen ein Client erhält. Beispielsweise kann es sinnvoll sein, einem Client keine Texturen zu übergeben. Der Multiplexer hält daher für jeden Client eine kleine Datenbank, in der Leistungsmerkmale und unterstützte Medien des Clients verwaltet werden.

I/O Plugins und Codecs sind dynamische ACE Dynamic Link Libraries, die zur Laufzeit geladen und wieder entfernt werden können. Jeweils ein eigener Manager kümmert sich um die Verwaltung der Plugins und Codecs. Die Codecs bieten ein einfaches Interface zum kodieren und dekodieren von zu übertragenden Daten. Die Plugins leisten ähnliches, beziehen sich aber auf ein Dateisystem.

Die Netzwerk-Komponente abstrahiert die Socket Verbindungen zwischen Client und Server. Das System beruht auf einem vereinfachten Nachrichtenkonzept, wie es von Agentenplattformen bekannt ist. Die Netzwerk-Kompontente arbeitet taskorientiert. Die übrigen Komponenten vergeben einen Auftrag beziehungsweise einen Task an die Netzwerk-Komponente, welche die Aufgaben an sogenannte Netzwerk-Agenten weiterleitet. Selbige begleiten einen Task bis zur dessen vollständigen Abarbeitung. Mittels der Tasks ist auch eine Synchronisation möglich, da der Auftraggeber auf die Ergebnisse eines Tasks warten kann. Mögliche Aufträge sind unter anderem das Versenden von Nachrichten oder der Aufbau einer Verbindung zwischen Client und Server.

Ähnlich der Netzwerk-Komponente arbeitet die I/O-Komponente ebenfalls eine Liste von Tasks ab. Allerdings gibt es hier nur zwei Threads. Ein Thread bearbeitet die Taskqueue und ein anderer Thread schreibt und liest die Daten in das beziehungsweise aus dem Dateisystem. Auf diese Weise wird ein Flattern der Köpfe verhindert, sofern es sich bei dem Medium um eine Festplatte handelt.

Ein Teil der Präsentation-Komponente auf Seite des Clients archiviert die Navigation des Benutzers und bestimmt hierüber die Dimensionen der Area-of-Interest. Diese wird in regelmäßigen Abständen an den Server übertragen, welcher sie an den Scheduler übergibt.

Das Occlusion Culling macht Gebrauch von Feedback Modus, der in OpenGL zum Standard gehört. Über diesen Modus ist es möglich, die an der Sichtpyramide geclippten Fragmente eines Primitive in Bildschirmkoordinaten zu erhalten. Der Zustand *Intern* kann über eine orthogonale Projektion mit Hilfe der Grafikhardware identifiziert werden. Die Front-to-Back Sortierung erfolgt über einen BSP Baum. Ein erweiterter Sutherland-Hodgman Algorithmus reduziert den dabei auftretenden Clippingaufwand.

Kapitel 8

Anwendungen und Ergebnisse

Dieses Kapitel widmet sich den Ergebnissen der Arbeit. Zuvor werden allerdings zwei Projekte vorgestellt, in denen das implementierte System zum Einsatz kommt.

8.1 Projekte

Das realisierte Systeme kommt derzeit in zwei verschiedenen Projekten zum Einsatz. Die grundsätzliche Anforderungen des Projektes *Huge Ubiquitous Graphical Objects* (HUGO) liegen in der Repräsentation, Übertragung und Visualisierung großer, interaktiver und dynamischer 3D Szenen, d.h. die Ziele sind deckungsgleich mit denen der vorliegenden Arbeit. Das Projekt HUGO ist von der Heinz Nixdorf Stiftung gefördert und wurde im Frühjahr 2004 erfolgreich abgeschlossen. Das zweite Projekt ist ein kommerzielles Produkt mit der Bezeichnung *In3D Viewer*, welches in Kooperation mit der Firma GIStec GmbH entwickelt wurde. Schwerpunkt hier ist die interaktive Darstellung und Übertragung großer Landschafts- und Städtemodelle. Anwendungsgebiete sind unter anderem Navigationshilfen und Touristikinformationssysteme. In beiden Projekten wird eine identische Implementierung verwendet, was für die Vielseitigkeit des Systemes zeugt. Allerdings sind in Hinsicht auf Landschafts- und Städtemodelle viele Spezialisierungen möglich, beispielsweise in der Repräsentation und Visualisierung von Höhengittern oder bei Visibility Culling Verfahren für Städte. Aus diesem Grund ist eine Konzentration des Systems auf *Geographical Information Systems* (GIS) geplant, welche in einem neuen Projekt *Transfer und Visualisierung Geographischer 3D Objekte* (TRAVO) realisiert werden soll. Dieses Projekt wird ebenfalls von der Heinz Nixdorf Stiftung gefördert.

Der In3D Viewer wurde zuerst von der Firma GIStec GmbH auf der INTERGEO 2002 präsentiert, welche weltweit die größte Fachmesse für den GIS Bereich darstellt. Die Vorführung bestand aus der Visualisierung des gesamten Bundeslandes Hessen mit einer Gesamtfläche von ca. 21500 Quadratkilometern. Die visualisierte Datenmenge basierte auf dem amtlichen Höhenmodell DGM25 des Bundeslandes Hessen mit einer Auflösung von 40 Metern und den offiziellen Satellitenbildern, welche als Texturen mit einer Auflösung von $2,5$ Metern vorhanden waren. Beide Datensätze wurden vom Hessischen Landesvermessungsamt (HLVA)

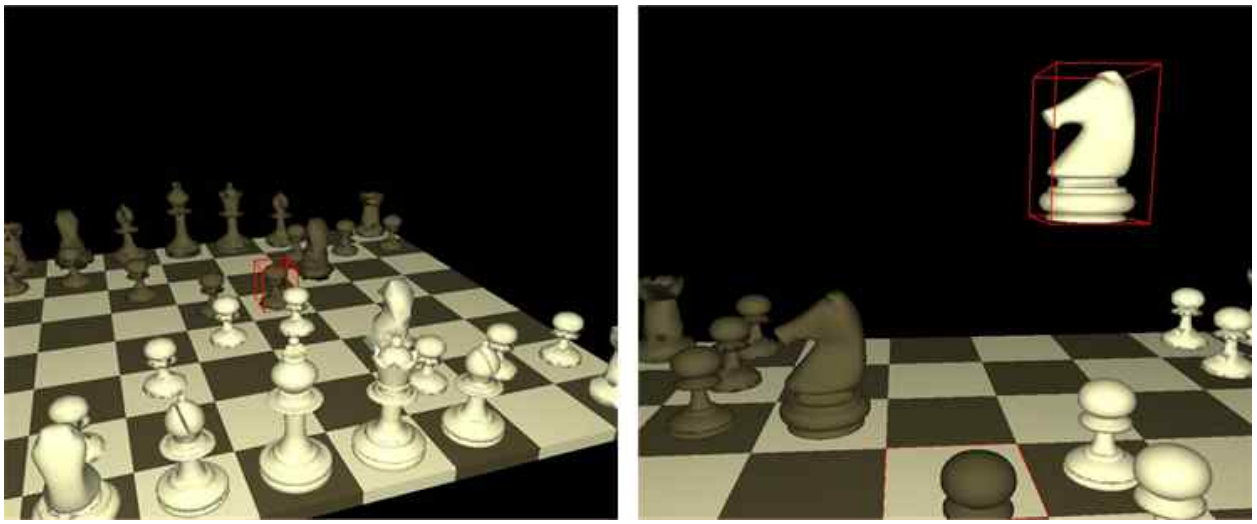

Abbildung 8.1: *Im Rahmen des HUGO Projektes wurde unter anderem ein interaktives Online Schachspiel implementiert.*

bereitgestellt. Zusätzlich beinhaltet die Visualisierung das Modell der Stadt Darmstadt mit etwa 22000 Gebäuden, welches die Deutschen Telekom zur Verfügung stellte.

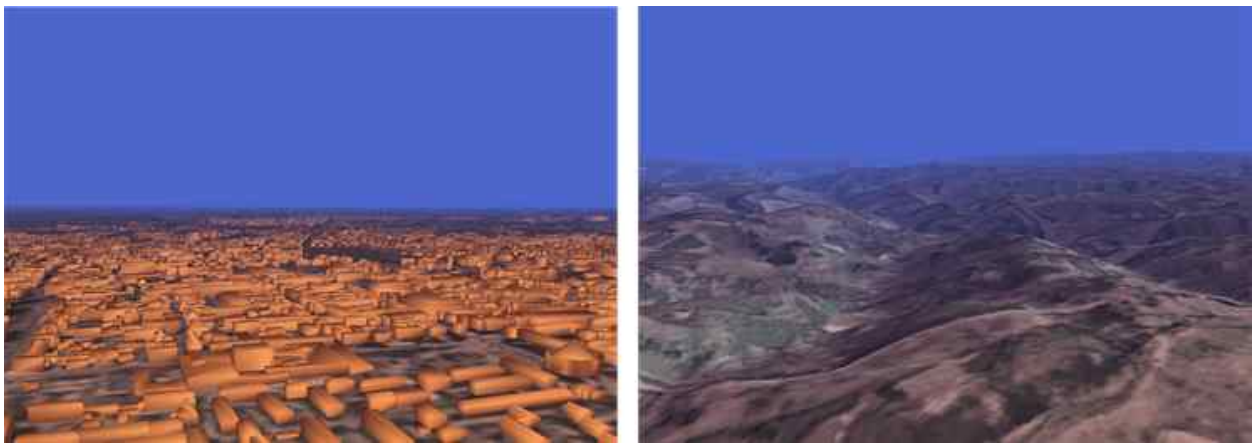

Abbildung 8.2: *Die Präsentation des Bundeslandes Hessen einschließlich der Stadt Darmstadt auf der INTERGEO 2002.*

Nach der Visualisierung des Bundeslandes Hessen folgten weitere Kooperationen, unter anderem mit dem Bundesland Thüringen sowie den Städten Wiesbaden und Hamburg. Hier kamen farbige Bodentexturen zum Einsatz, was die Datenmenge gegenüber der Visualisierung des Bundeslandes Hessen deutlich erhöhte. An dieser Stelle zahlte sich die Verwendung des in Abschnitt 7.1.6 eingeführten progressiven Texturformats aus. Erfolgten die Visualisierungen der Städte Wiesbaden und Hamburg noch ohne Texturen der Häuser, so gab es für die Darstellung der Stadt Neubrandenburg fotografierte Bilder der Straßenzüge und Gebäude.

Während die Geometrien der Häuser in der Regel durch Software automatisch aus den Grundrissen erzeugt werden, ist das Erzeugen der Texturen der Häuser immer noch ein sehr aufwändiges Unterfangen, da die Bilder von Hand fotografiert werden müssen. Die Darstellung der üblichen Daten im GIS Bereich ist immer noch sehr schwierig, weil die generierten

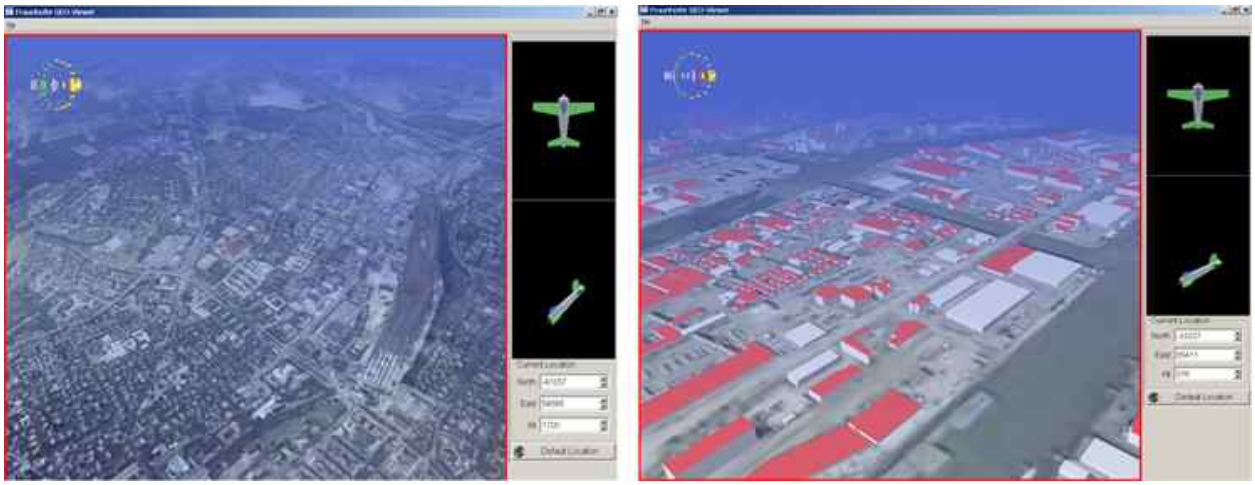

Abbildung 8.3: *Die Visualisierung von Wiesbaden und Hamburg.*

Informationen sehr fehlerbehaftet sind. Beispielsweise sind die Geometrien der Gebäude nicht einheitlich orientiert, was zu fehlerhaften Normalen und somit etwa zu falschen Lichtberechnungen führt oder gar ein einfaches Backface Culling unmöglich macht. Oftmals sind auch keine geeigneten Hierarchien der Gebäude vorhanden. Besteht ein Gebäude aus mehreren Objekten wie beispielsweise Säulen am Eingang und dem eigentlichen Hauptteil, so liegen die entsprechenden Objekte auf der gleichen weil einzigen Hierarchieebene. Für die Übertragung bedeutet dies, dass für alle Objekte das vollständige Protokoll gefahren werden muss und somit mit einem einzelnen Gebäude sehr viel Kommunikationsaufwand entstehen kann. Die Texturen schließen häufig nicht bündig ab, d.h. Teile der Wände sind an gemeinsamen Ecken verzerrt oder Bereiche des Bürgersteiges sind Bestandteil der Häuserwand. Aufgrund der gewaltigen Datenmengen ist aber eine Verbesserung der Qualität äußerst problematisch, da üblicherweise die Informationen von Hand zu überarbeiten sind.

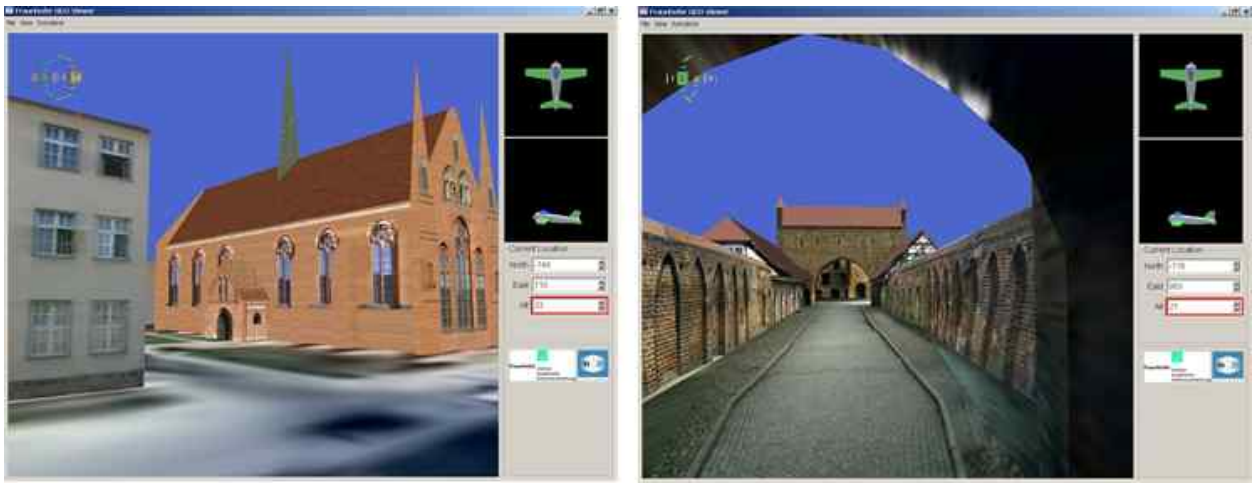

Abbildung 8.4: *Für die Stadt Neubrandenburg standen auch Texturen der Häuser zur Verfügung.*

8.2 Ergebnisse der räumlichen Sortierung

Für das Testen des in Abschnitt 6.3.4 beschriebenen Konzepts der räumlichen Sortierung wurden Ergebnisse aus Partikelsystemen verwendet. Die Darstellung der Partikel erfolgte dabei wie in Abbildung 8.5 über Würfel. Geschwindigkeit und Richtung der Würfel wurden durch die zugrunde liegende Simulation bestimmt. Als Testsystem diente ein P4 mit 2400 Mhz und 1 GB Speicher. Für die im Folgenden beschriebenen Tests ist der Begriff der Dichte δ_E von Bedeutung. Selbige beschreibt das Verhältnis des Gesamtvolumens aller in der Szene vorhandenen Elemente zum Volumen, welches durch die Wurzel des Raumunterteilungsbaumes definiert wird. Je höher die Dichte, desto wahrscheinlicher werden Schnittelemente. Während der Tests wurde die Anzahl der Partikel von 100000 auf 500000 gesteigert. Im Maximalfall lag die Dichte bei ca. 6%. d.h. die Elemente belegten 6% des Szenenraumes. Abbildung 8.5 zeigt eine Szene bei einer Dichte von etwa 0,5%. Um einen Vergleich bieten zu können, kamen während der Tests drei verschiedene Raumunterteilungsbäume zum Einsatz: Ein normaler Octree, ein optimierter Octree, der über das Konzept der ausgleichenden Bewegungen verfügte, sowie der im Konzept vorgschlagene Baum mit 27 Kindknoten, im Folgenden *Tree27* genannt.

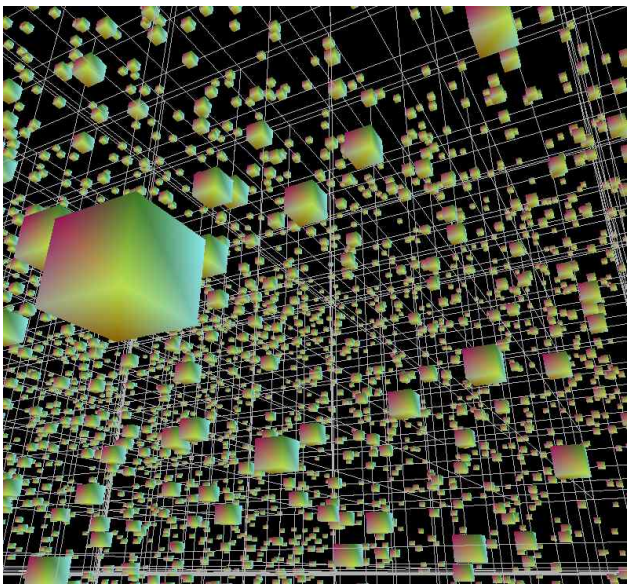

Abbildung 8.5: *Für das Testen wurden Simulationsergebnisse von Partikelsystemen verwendet.*

Abbildung 8.6 zeigt die Ergebnisse der *move* Operation anhand der drei Baumarten. Der normale Octree schneidet dabei deutlich am schlechtesten ab, da er kompensierende Beweungen nicht berücksichtigt. Stattdessen kann sich jedes Ein- beziehungsweise Ausfügen eines Elements direkt auf die Struktur des Raumunterteilungsbaumes auswirken. Im Gegensatz zu den beiden anderen Bäumen unterscheidet der normale Octree nicht zwischen einer Transformationsphase und einer Reorganisationsphase, d.h. die Struktur des normalen Octrees ist immer auf dem aktuellen Stand. Aus diesem Grund sind die Ergebnisse aus Abbildung 8.7 noch zu den jeweiligen Ergebnissen aus Abbildung 8.6 auf zu addieren. Aber selbst dann ist der normale Octree immer noch deutlich langsamer.

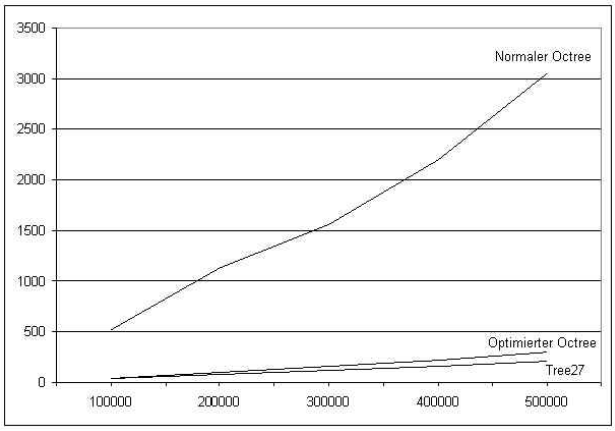

Abbildung 8.6: *Die Ergebnisse der move Operation. Die y-Achse beschreibt die Zeit in Millisekunden, die x-Achse die Anzahl der Elemente in der Szene.*

Der *Tree27* ist in der Transformationsphase etwas schneller als der optimierte Octree. Die Ursache liegt im Finden des neuen Zielknotens bei der Bewegung eines Elementes. Hier gibt es grundsätzlich zwei verschiedene Möglichkeiten: Die erste Variante sucht den neuen inneren Zielknoten immer ausgehend von der Wurzel, steigt also stets den Raumunterteiungsbaum hinab. Die zweite Variante klettert vom momentanen Vaterknoten des Elementes aufwärts bis ein Vaterknoten erreicht wird, der die AABB des Elements vollständig umschließt. Anschließend versucht der Ansatz, das Element möglichst tief in einen der Teilbäume des gefundenen Vaterknotens einzuordnen. Hier kann also bei der Suche ein Richtungswechsel auftreten. Im System kommt die zweite Variante zum Einsatz, was sich besonders für den *Tree27* bezahlt macht. Hier ist die Struktur deutlich feiner als beim optimierten Octree, weshalb bei einer gleichmäßigen Bewegung das Finden des Zielknotens ein ganzes Stück schneller erfolgt. Anders formuliert, die Wahrscheinlichkeit, dass der zu findende benachbarte innere Knoten in der Baumstruktur den gleichen direkten Vaterknoten wie der Quellenknoten hat, ist wesentlich höher. Interessanterweise ist ürigens nicht die Reorganisation des Baumes der Flachenhals bei der Repräsentation dynamischer Szenen, sondern eben die Berechnung der Zielknoten. Reorganisation und Suche stehen etwa im Verhältnis 1 zu 3 bis 1 zu 2.

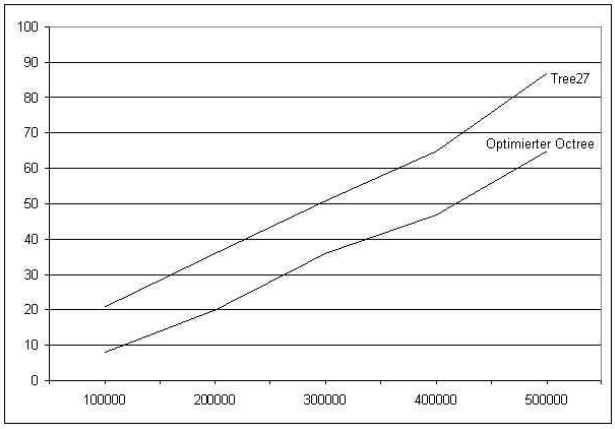

Abbildung 8.7: *Die Ergebnisse der Reorganisation. Die y-Achse beschreibt die Zeit in Millisekunden, die x-Achse die Anzahl der Elemente in der Szene.*

Bei der in Abbildung 8.7 dargestellten Reorganisation schneidet der optimierte Octree dagegen etwas besser als der *Tree27* ab. Das ist nicht verwunderlich, da die Struktur des optimierten Octrees eine geringere Komplexität als der *Tree27* aufweist und über weniger innere Knoten verfügt. Der normale Octree kommt hier nicht zur Geltung, weil hier keine Reorganisation zum Einsatz kommt. Neben der *move* Operation gibt es noch die *add* und *remove* Operationen, welche das Ein- beziehungsweise Ausfügen eines Elements realisieren. Auch dort schneidet der optimierte Octree aufgrund seiner einfacheren Struktur geringfügig schneller ab. Auf dem Testsystem war innerhalb einer Sekunde das Einfügen von etwa 150000 Elementen in einen leeren *Tree27* einschließlich der Reorganisation möglich. Das Entfernen der Elemente ist etwas aufwändiger, da Teile des Baumes und die Elemente selbst zu löschen sind. Hier waren etwa 120000 Operationen möglich. Das Transformieren der Elemente plus der Reorganisation beansprucht bei 500000 Elementen etwa 330 ms. Der Vorgang ist also dreimal pro Sekunde möglich. Im Vergleich benötigt der normale Octree schon für einen Vorgang mehr als drei Sekunden. Durch den Einsatz der Temporary Bounding Boxes fällt die Berechnung des Zielknotens weg, wodurch deutlich niedrigere Laufzeiten möglich sind. Entsprechende Tests brachten aber keine neuen Erkenntnisse, weil einfach bestimmte Bereiche des Raumunterteilungsbaumes statisch blieben, die betroffenen Elemente also nicht als dynamisch anzusehen waren.

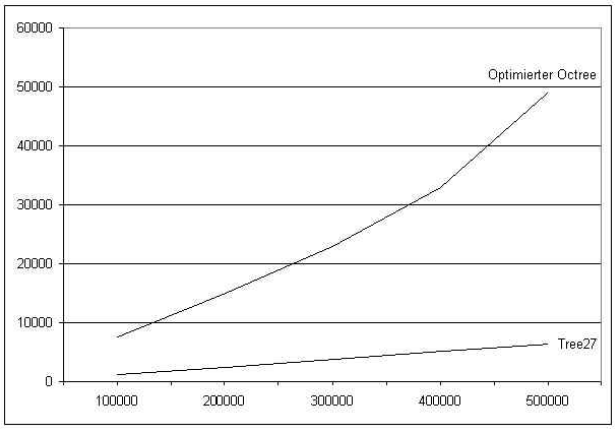

Abbildung 8.8: *Die Ergebnisse bei einer Beispielanwendung, in diesem Fall einem Visibility Culling. Die y-Achse beschreibt die Anzahl der vom Verfahren bearbeiteten Elemente, die x-Achse die Anzahl der Elemente in der Szene.*

Wirklich interessant wird die Effizienz der Bäume aber erst bei einer Beispielanwendung. Durch die schlechte Behandlung der Schnittelemente im Falle des optimierten Octrees ist das durchschnittliche Level der Elemente deutlich höher als beim *Tree27*. Algorithmen wie etwa ein Visibility Culling sind hierdurch ineffizienter. Abbildung 8.8 zeigt die Ergebnisse für ein entsprechendes Verfahren anhand des optimierten Octrees und des *Tree27*. Hier schneidet der *Tree27* sehr viel besser ab, da das Visibility Culling wesentlich weniger Elemente berücksichtigen muss. Die Sichtweite lag während der Tests bei 5000 Metern, wobei die Szene selbst einem Würfel mit der Kantenlänge von 8000 Metern entsprach.

8.3 Ergebnisse der progressiven Simplifizierung

Mit der Verwendung eines progressiven Simplifizierungsverfahren für die automatische Generierung der Level-of-Detail ist leider das Problem verbunden, qualitativ hochwertige Reduktionen nur bei Modellen mit hoher Polygonzahl und möglichst sanften Rundungen erzielen zu können. Allerdings ist dies ein allgemeines Problem automatischer Level-of-Detail Verfahren. Abhilfe schafft in der Regel nur die Nachbearbeitung der Figuren von Hand, wie es vor allem in der Spiele Industrie der Fall ist. Eine weitere Möglichkeit, Geometrie bei hoher visueller Qualität einzusparen, liegt im Bumpmapping. Die Normalmaps werden hier mittels des Orignalmodelles erzeugt. Anschließend wird das Modell stark simplifiziert und darauf die Normalmaps sowie Farbtexturen gelegt. Jedoch ist auch hier eine Nachbearbeitung durch den Designer notwendig.

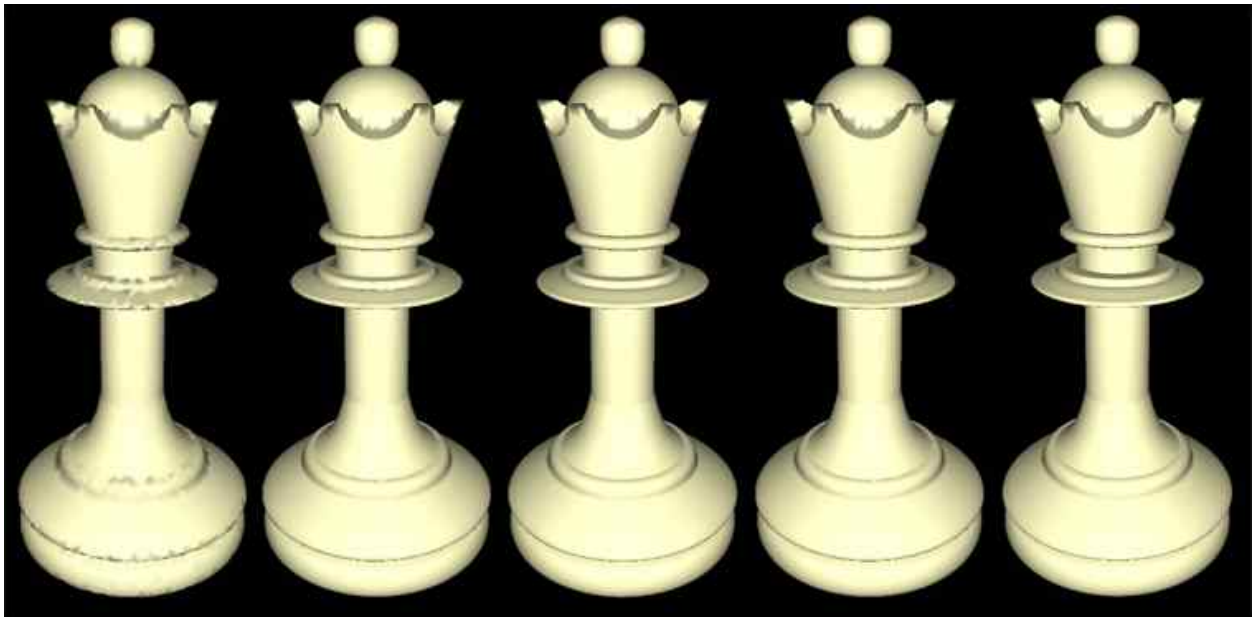

Abbildung 8.9: *Eine simplifizierte Schachfigur mit Normalen und Farben in 10%, 30%, 50%, 75% und 100% Qualität.*

Abbildung 8.9 zeigt eine Schachfigur in fünf verschiedenen Auflösungen, die durch den in Abschnitt 6.4 erzeugt wurden. Die Figur ganz rechts entspricht dem Original und besteht aus ca. 90000 Dreiecken. Aufgrund der relativ flachen Krümmungen zwischen den benachbarten Dreiecken wirkt sich die Simplifizierung erst bei der Figur ganz links deutlich aus, welche ca. 9000 Dreiecke beansprucht.

Für ein 3D Spiel, in dem eine menschliche Figur derzeit üblicherweise um die 2000 Dreiecke besitzt, sind 9000 Dreiecke jedoch immer noch zuviel. Die Ursache für den Artefakte der linken Dame liegen in der fehlenden Berücksichtigung der Normalen durch die Simplifizierung. Im Falle der Damen sind die Normalen per Vertex definiert und werden lediglich im Sinne des progressiven Formats vertauscht. Durch die Entfernung von Dreiecken ergeben sich aber neue Normalen für die Scheitelpunkte. Eine Lösung besteht in der Aktualisierung der Normalen, was aber zusätzlich im progressiven Format vermerkt werden muss und insofern mehr Speicher benötigt. Eine weitere Option liegt darin, die Normalen nicht zu berücksichtigen

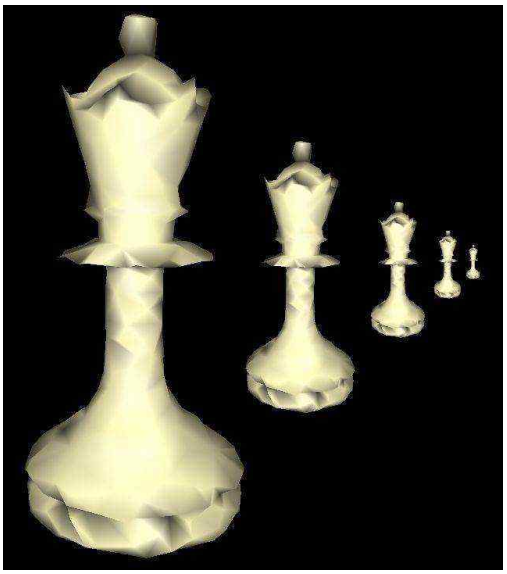

Abbildung 8.10: *Die Dame mit nur noch etwa 150 Dreiecken. Obwohl die Figur aus der Nähe aufgrund der nicht neu berechneten Normalen deutliche Artefakte aufweist, ist das Resultat aber für eine weiter entfernte Figur als niedriges Level-of-Detail sehr gut verwendbar.*

und nur bei Bedarf zu generieren. Hierdurch wird Speicher und Übertragungszeit gespart, allerdings ein höherer Rechenaufwand erforderlich. In vielen Fällen kann die Hardware weiterhelfen. Abbildung 8.10 zeigt die Dame mit noch etwa 150 Dreiecken, worauf deutliche Artefakte vor allem an scharfen Kanten zu sehen sind. Für nahe Level-of-Detail ist die Qualität ungenügend, in größeren Entfernungen aber vollkommen ausreichend. Zudem kann die Figur trotz der extrem niedrigen Auflösung auch noch als Dame eines Schachspiels erkannt werden.

Abbildung 8.11 stellt das gesamte Schachspiel dar, wobei das linke Bild auf einem Laptop mit einem AMD 1000 MHz, 512 MB Speicher und einer GeForce 2 Grafikkarte und das rechte Bild auf einem Standard PC mit einem P4 2400 MHz, 1 GB Speicher und einer ATI Radeon 9800 aufgenommen wurde. Der Laptop war mit einem mobilen 11 Mbit WLAN und der PC mit einem Festnetz verbunden. Daraus resultierten auf dem Laptop eine Szene mit etwa 10% und auf dem PC eine Szene mit etwa 80% der originalen Qualität. In Abhängigkeit von diesen Werten erzeugten die Clients fünf Level-of-Details, wss insbesondere auf dem linken Bild zu sehen ist. Die wei0en Figuren im Hintergrund verfügen über etwa 8 bis 10 Polygone und sind trotzdem noch in ihrer Funktion zu erkennen. Die Einstellung der Level-of-Detail wurde zu Demonstrationszwecken bewusst so gewählt, dass auch in geringer Nähe zum Betrachter niedrige Detailstufen angezeigt wurden. Die entsprechenden Schwellwerte lassen sich bei Bedarf ändern.

Zur Bestimmung der darzustellenden Detailstufe bietet das System mittels des Konzepts der Renderer mehrere Lösungen an. Die einfachste Form der Berechnung besteht im Abstand zum Betrachter. Eine andere Variante existiert über die NVIDIA-Occlusion-Query, welche aber nur von aktuelleren Grafikchips unterstützt wird. Hier berechnet der Renderer die Anzahl der Pixel, die von der Bounding Box eines Elementes sichtbar sind und bestimmt daraus die Detailstufe. Letzterer Ansatz ergibt im Durchschnitt bessere Ergebnisse, hat aber den

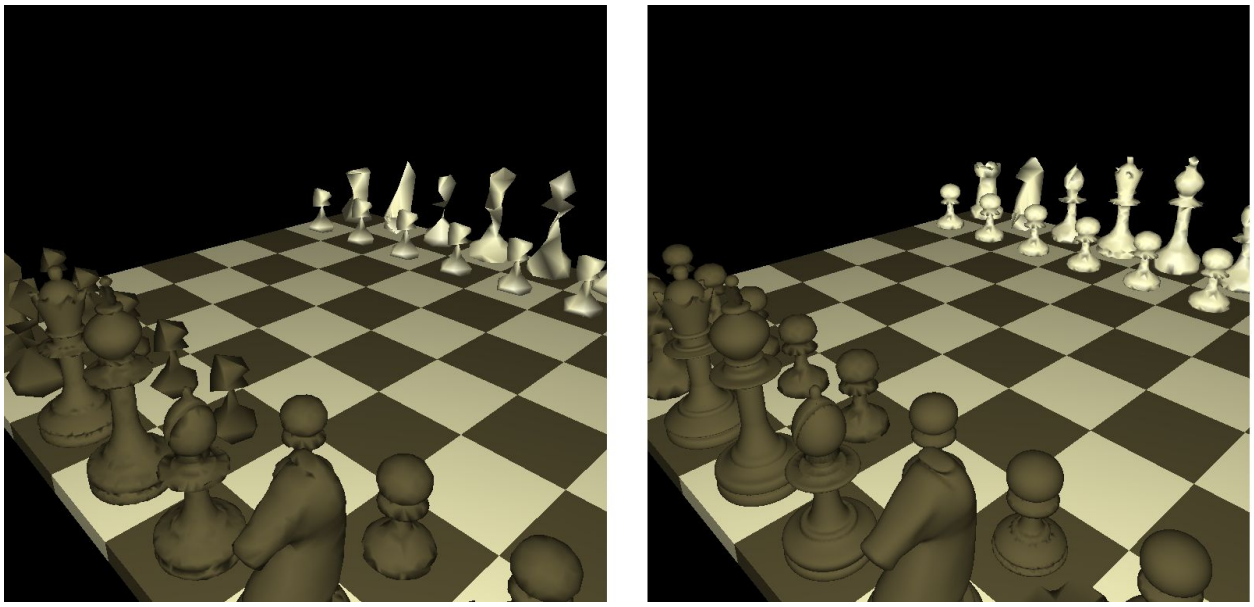

Abbildung 8.11: *Während das linke Bild das Schachspiel mit einer Qualität von ca. 10% auf einem Laptop darstellt, zeigt das rechte Bild die gleiche Szene mit ca. 80% Qualität auf einem Standard PC.*

Nachteil, dass in der Nähe gelegene Objekte mit geringem Pixelanteil in niedriger Auflösung dargestellt werden. In der Regel handelt es sich dabei um Elemente, die lediglich am Rand des Sichtfeldes zu sehen sind, also nicht im Fokus des Betrachters stehen. Abbildung 8.12 zeigt eine derartige Situation am Beispiel einer Visualisierung der Stadt Wiesbaden. Das Höhenfeld ist in mehrere Kacheln unterteilt, welche jeweils mit einer Textur belegt sind. Im linken Bild sind zwei benachbarte Kacheln zu sehen, von denen eine am Rand des Sichtfeldes liegt. Der weiße Kreis markiert die Nahtstelle zwischen den beiden Kacheln. Die Auflösung der am Rand verlaufenden Kachel ist deutlich niedriger als der benachbarten Kachel, da erstere trotz nahezu identischer Entfernung wesentlich weniger Pixel beansprucht. Die Nahtstelle ist somit sehr markant. Im rechten Bild haben dagegen beide Kacheln einen hohen Pixelanteil, weshalb die Auflösung der Texturen identisch ist und die Nahtstelle sehr viel undeutlicher verläuft. Dass der Übergang immer noch zu sehen ist, hängt mit einem weiteren Problemen im GIS Bereich zusammen: Die Kacheln werden in der Regel zu unterschiedlichen Tageszeiten beziehungsweise bei unterschiedlichen Lichtverhältnissen aufgenommen.

Die im GIS Bereich üblichen Nachbildungen der Gebäude verfügen mit Ausnahme einiger Sehenswürdigkeiten nur über eine geringe Anzahl von Polygonen. Automatische Simplifizierungsverfahren machen hier daher wenig Sinn. Hier sind eher Ansätze mittels Imposter oder spezielle Vereinfachungen wie das Verwenden der Bounding Box als niedrigste Detailstufe angesagt. Abbildung 8.13 zeigt nochmals ein komplexeres Modell mit ca. 39000 Dreiecken im Original. Im Unterschied zur Dame handelt es sich hier um eine Elementhierarchie, da die Augen eigene Elemente darstellen.

Die implementierte Simplifizierung wurde hinsichtlich der Geschwindigkeit mit dem Ansatz von Schröder verglichen, da sein Verfahren hier ein Preferenz darstellt. Die von Schröder bekannte Turbine Blade mit ca. 1,7 Millionen Dreiecken, Farben und Normalen benötigt auf einem P4 mit 2400 MHz etwa 120 Sekunden für die Simplifizierung einschließlich der

Abbildung 8.12: *Die Bestimmung der Detailstufe über die Anzahl der Pixel kann wie im linken Beispiel bei am Rand gelegenen Elementen zu Artefakten führen. Im rechten Bild hängen die sichtbaren Artefakte dagegen nur noch mit den Originaldaten zusammen.*

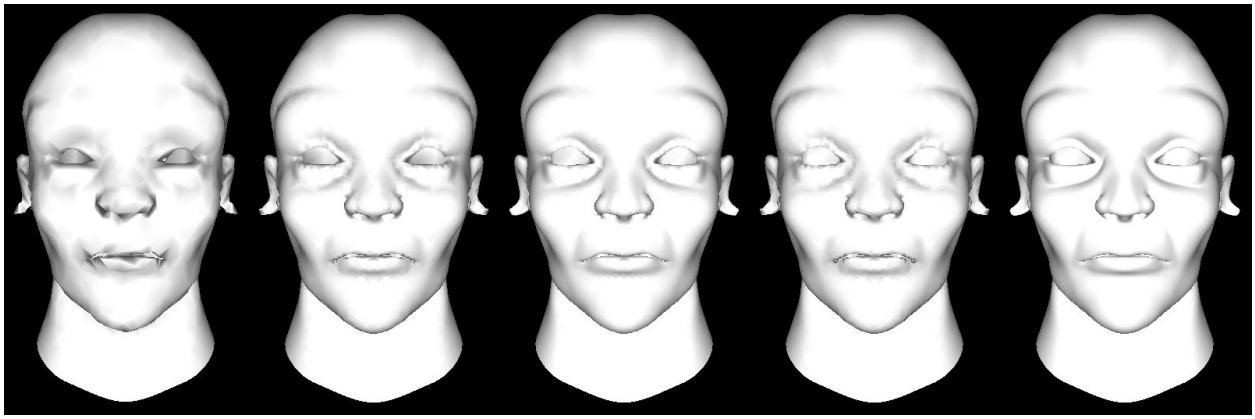

Abbildung 8.13: *Ein simplifizierter Elfenkopf ebenfalls in 10%, 30%, 50%, 75% und 100% Qualität. Der Kopf besteht aus einer Elementhierarchie, da die Augen durch eigene Elemente repräsentiert sind.*

Erzeugung des progressiven Formats. Der Ansatz von Schröder kommt ohne Farben und Normalen etwa auf 130 Sekunden. Der bekannte Standford Bunny mit etwa 66000 Dreiecken veranschlagt bei Einsatz des vorgestellten Verfahrens auf dem gleichen System knapp 2 Sekunden.

8.4 Ergebnisse der Übertragung

Der Aufwand der Übertragung wird hauptsächlich durch die Anzahl der selektierten Elemente verursacht. Hierfür zeichnen sich zum einen der in Abschnitt 6.5 beschriebene Ansatz der Area-of-Interest und zum anderen die Anpassung der Zonen an die Navigation aus Abschnitt 6.7.1 verantwortlich. Die Testreihen erfolgten wie schon im vorangehenden Ansatz auf einem P4 mit 2400 MHz und einem GB Speicher.

Obgleich der Scheduler effizienter in einer Szene mit dreidimensionaler Verteilung der Ele-

mente arbeitet, wurden für die Tests Szenen mit nahezu planarer Verteilung gewählt. Derartige Szenarien lassen sich häufig in Stadt- und Landschaftsmodellen finden. Die maximale Verteilung umfasste in x und z Richtung etwa 65000 Meter und in y Richtung zirka 600 Meter. Die Ergebnisse des Schedulers werden hauptsächlich von der Anzahl der Elemente und der Menge der registrierten Clients beeinflusst. Ein wichtiger Wert ist hier wie schon bei der räumlichen Sortierung die Dichte δ_E, welche das Verhältnis zwischen dem Gesamtvolumen der Elemente und dem durch die Wurzel des Raumunterteilungsbaumes umschlossenen Volumens beschreibt. Während sämtlicher Tests lag δ_E auf einem nahezu konstantem Wert von ca. 60%, selbst wenn die Anzahl der Elemente verändert wurde [1]. Um diesen konstanten Wert zu erhalten, varierte die Verteilung der Elemente in x und y Richtung von 22000 Meter bis zu 65000 Meter. Ein weiterer wichtiger Werte ist die Dichte δ_C. Sie repräsentiert das Verhältnis zwischen dem Gesamtvolumen aller Areas-of-Interest und dem Gesamtvolumen der Szene. In den Tests wurde δ_C linear von 1% bis auf 60% gesteigert, was einer Steigerung der Clientzahl von 10 auf 400 entsprach.

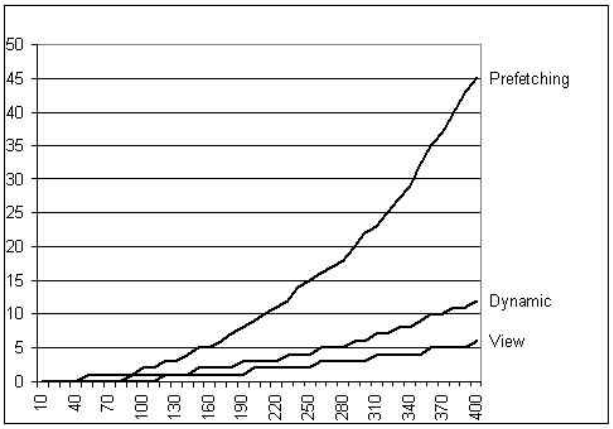

Abbildung 8.14: *Die Ergebnisse für das Einfügen der Clients in den jeweiligen Clientbaum. Die x-Achse des Diagrammes repräsentiert die Anzahl der Clients, die y-Achse die gemessene Zeit in Millisekunden.*

Abbildung 8.14 illustriert das Laufzeitverhalten beim Einfügen der Clients in die drei Clientbäume. Demnach ist das Einfügen der präventiven Zonen deutlich teurer als das der dynamischen und sichtbaren Zonen. Dies hängt unmittelbar mit der Größe der einzelnen Zonen zusammen. Die präventiven Zonen sind im Durchschnitt wesentlich größer als die dynamischen Zonen und diese wiederum als die sichtbaren Zonen. Mit wachsender Ausdehnung der Zonen steigt die Wahrscheinlichkeit der Überlappung der Zonen, wodurch es beim Einfügen zu Schnittberechnungen und somit zu einer deutlich höheren Komplexität des betroffenen Clientbaumes kommt. Einen erheblichen Einfluss auf die Struktur eines Clientbaumes hat die Reihenfolge der Clients beziehungsweise deren Lage zueinander während dem Einfügen. Im schlimmsten Fall liegen die Clients nacheinander auf einer diagonalen Linie, was zu einem extrem unbalancierten Baum führt. Die Komplexität der Einfügeoperation liegt dann bei $O(n)$. Im besten Fall des balancierten Baumes liegt die Komplexität bei $O(log(n))$. Hierzu müssen die Clients jedoch in alle drei Dimensionen sortiert werden, so dass eine Zone beim Einfügen möglichst immer einen neuen Halbraum erzeugt. Abbildung 8.14 beschreibt das

[1] Die Elemente beanspruchten also trotz der Würfelform der Wurzel des Raumunterteilungsbaumes und trotz der nahezu planaren Verteilung etwa 60% der durch den Raumunterteilungsbaum umschlossenen Szene.

durchschnittliche Verhalten bei 1000 Versuchen mit zufälliger Reihenfolge der Clients. Die Tests ergaben, dass eine Vorsortierung der Clients gegenüber der zufälligen Variante keine Laufzeitverbesserung erbrachte und zwar nicht nur hinsichtlich dem Einfügen sondern auch im Bezug auf die spätere Auswertung des Baumes. Aus Abbildung 8.14 geht außerdem hervor, dass die Registrierung von 100 Clients etwa 3 ms dauert, d.h. der Aufbau des Baumes erfolgt sehr schnell.

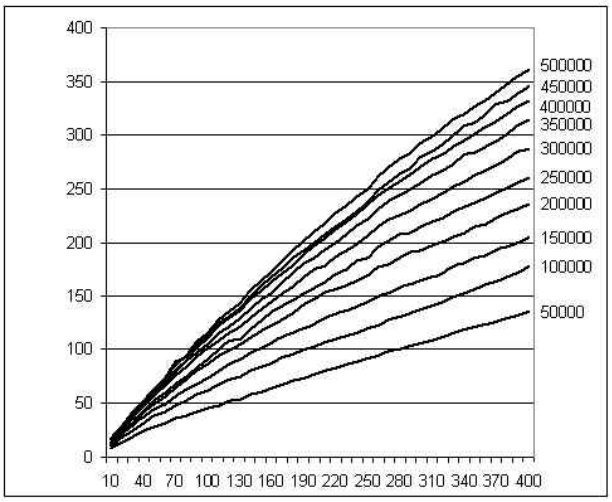

Abbildung 8.15: *Die Auswertung des Clientbaumes mit den präventiven Zonen. Die x-Achse beschreibt die Anzahl der Clients, die y-Achse die benötigte Zeit in Millisekunden. Hinter jeder Kurve steht die in der Szene vorhandene Anzahl der Elemente.*

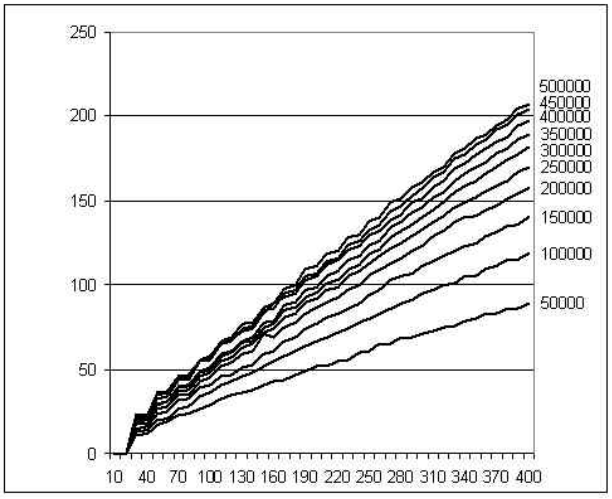

Abbildung 8.16: *Die Auswertung des Clientbaumes mit den dynamischen Zonen.*

Die Abbildungen 8.15, 8.16 und 8.17 beschreiben das Testen der Elemente gegen die jeweiligen Clientbäume. Wichtig ist hier, dass die Laufzeit weder proportional zur Anzahl der Clients noch zur Anzahl der Elemente liegt. Aufgrund der konstanten Dichte δ_E steigen die Kurven allerdings gegen Ende nahezu linear. Mit jedem neuen Client gibt es dann eine bestimmte, ebenfalls nahezu konstante Anzahl von Elementen innerhalb der Area-of-Interest

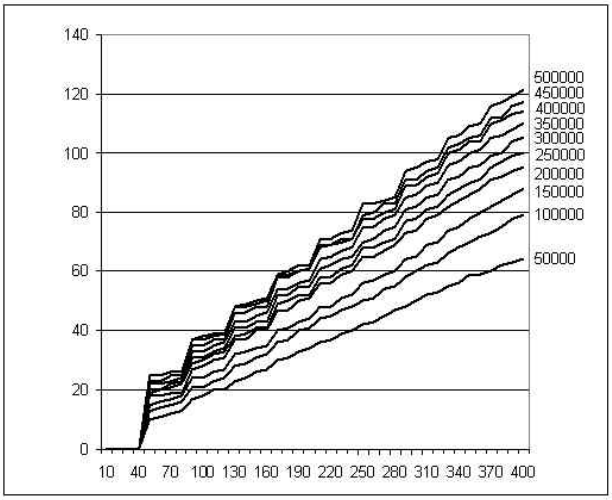

Abbildung 8.17: *Die Auswertung des Clientbaumes mit den sichtbaren Zonen..*

des Clients, welche selbigem zugewiesen werden muss. Die Auswertung des Clientbaumes mit den präventiven Zonen verlangt bei 100 Clients und 500000 Elementen etwa 110 ms. Da die Elemente innerhalb der anderen Zonen jeweils nur eine Untermenge der präventiven beziehungsweise dynamischen Zonen darstellen, fallen die in Abbildung 8.16 und 8.17 aufgeführten Laufzeiten deutlich geringer aus als in Abbildung 8.15. Der Algorithmus benötigt für die dynamischen Zonen etwa 55 ms und die sichtbaren Zonen etwa 40 ms. Entscheidend ist, dass die Auswertung der höher priorisierten Zonen deutlich schneller als die der niedriger eingestuften Zonen möglich ist. Änderungen innerhalb der höher priorisierten Zonen müssen schneller erfasst und die betreffenden Zonen daher auch häufiger getestet werden. Gemäß dem Konzept der Zyklen aus Abschnitt 6.5.4 benötigt ein Zyklus ca. $110+2*55+4*44 = 380$ ms für 100 Clients und 500000 Elemente, sofern die dynamische Zone pro Zyklus zweimal und die sichtbare Zone viermal pro Zyklus getestet wird. Eine realistische Wiederholrate der Zyklen liegt bei 1 bis 5 Sekunden. Die Zeitspanne zwischen den einzelnen Zyklen benutzt der Server für die Übertragung der Elemente.

Einen großen Teil der Rechenleistung verschlingt die Komprimierung der Daten, welche der Multiplexer an die Clients versendet. Eine Komprimierung der progressiven Daten vor der eigentlichen Laufzeit des Servers, also beispielsweise ein bereits berechnetes Dateiformat auf einem externen Speichermedium, ist nur bedingt möglich, da jeder Client eine unterschiedliche Menge an Daten benötigt. Je nach Netzwerkverbindung ist es manchmal sogar sinnvoll, lediglich eine geringe oder eventuell gar keine Komprimierung vorzunehmen, da der Transfer ohne Komprimierung bei großer Bandbreite oftmals schneller ist als der Transfer einschließlich Komprimierung. Dies gilt vor allem für Modelle wie etwa einfache Gebäude, die nur eine kleine Datenmenge beanspruchen. Das progressive Format der in Abbildung 8.9 dargestellten Dame beansprucht mit Farben und Normalen etwa 4500 Kb. Eine normale Kodierung der gesamten Daten mittels eines zlib Codecs[2] kommt immerhin noch auf 2300 Kb und dauert etwa 300 ms. Die maximale Komprimierung spart sage und schreibe etwa 1 Kb, dauert aber etwa zehnmal so lange. Allerdings ist hierbei anzumerken, dass die zlib für

[2]Die zlib Bibliothek [ZIP] steht frei zur Verfügung und wurde in einen Codec integriert.

Fließkommazahlenwerte nur bedingt geeignet ist.

Bei niedrigen Bandbreiten und gleichzeitig hoher Komplexität der Modelle wird ein Standard PC als Server durch die Komprimierung bei etwa 10 Clients überlastet. Hier besteht allerdings die Möglichkeit einer Serverhierarchie, wie sie von Funkhouser vorgschlagen wird. Der Vorteil von diesem System ist, dass es einfach zu realisieren ist und enorme Geschwindigkeitsvorteile bringt. Die Ursache hierfür beruht darauf, dass der Hauptserver lediglich eine kleine Information an den Nachrichtenserver senden muss, ein bestimmtes Modell in der gewünschten Kompressionsstufe an einen Interessenten zu verschicken. Der Aufwand hierfür gegenüber der eigentlichen Komprimierung ist vergleichsweise gering.

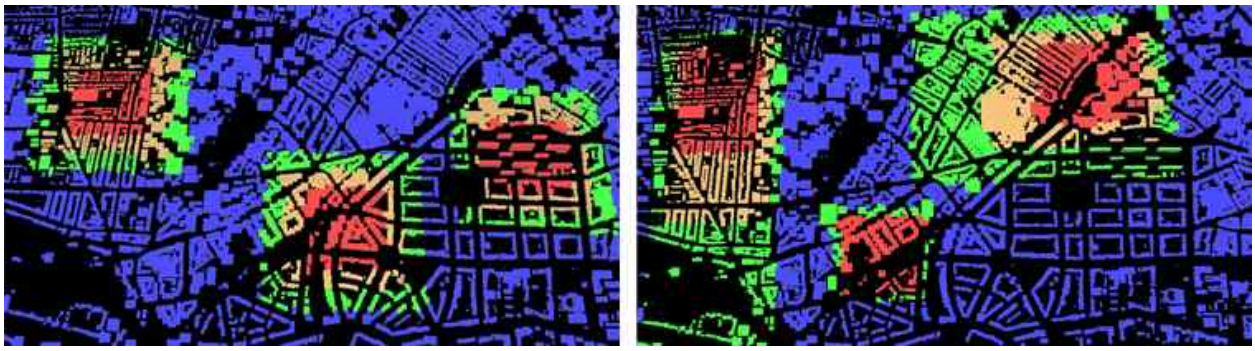

Abbildung 8.18: *Zwei Screenshots der Visualisierung auf Seiten des Servers. Drei Clients begehen die Innenstadt von Hamburg, deren Gebäude über ihre AABB repräsentiert sind.*

Für die Ergebnisse der Adaption der Area-of-Interest an die Navigation des Benutzers ist der Begriff der Verschwendungsrate von Bedeutung. Hierunter ist der Anzahl der Elemente einer Szene zu verstehen, die durch den Multiplexer zur Übertragung an einen Client selektiert aber dort niemals angezeigt werden. Abbildung 8.18 zeigt eine Visualisierung der Stadt Hamburg auf Seite des Servers. Dort sind die Elemente lediglich als AABB repräsentiert. Auf der linken Seite der Abbildung melden sich gerade drei Clients beim Server an. Die rechte Seite zeigt zwei Clients in Bewegung, wohingegen der dritte bereits längere Zeit auf der Stelle verharrt. Nach den Ergebnissen des Feldtests aus Abschnitt 6.7.1 gibt es drei grundlegende Bewegungsformen, nämlich Bewegungen auf der Stelle, sowie geradlinige und kurvenförmige Bewegungen. Ohne eine Adaption liegt die Verschwendungsrate bei den im Feldtest gestellten Aufgaben für Bewegungen auf der Stelle bei 25%, für geradlinige Bewegungen bei 28% und für kurvenförmige Bewegungen bei 33%. Durch den Einsatz der Adaption singt die Verschwendungsrate bei Bewegungen auf der Stelle auf 10%, bei geradlinigen Bewegungen auf 12% und bei kurvenförmige Bewegungen auf 18%. Dies deckt sich mit der Erkenntnis, dass Kurvenläufe die komplexeste Bewegungsform darstellen und nur sher schwer vorherzusagen sind. Für eine genaue Schilderung des Feldtests einschließlich seiner Ergebnisse sei auf [Hei04] verwiesen.

8.5 Ergebnisse der Visualisierung

Das in Abschnitt 6.7.2 vorgestellte Occlusion Culling wurde ebenfalls auf einem P4 mit 2400 MHz getestet. Der PC verfügte zudem über eine ATI RADEON 9800 Grafikkarte.

Der Algorithmus identifiziert nicht einzelne Polygone, sondern bestimmt die Sichtbarkeit von Animationsagenten. Daher wurde als Szenario ein Stadtmodell verwendet, wobei eine Steigerung der Gebäudeanzahl von 100000 auf 500000 erfolgte. Die Gebäude bestanden im Durchschnitt aus ca. 250 Dreiecken, d.h. die gesamte Szene umfasste maximal etwa 125 Millionen Dreiecke. Die Sichtweite war auf 30000 Meter definiert. Da die verwendete Szene 16 Quadratkilometer umfasste, war der Benutzer somit in der Lage, die gesamte Szene zu überblicken.

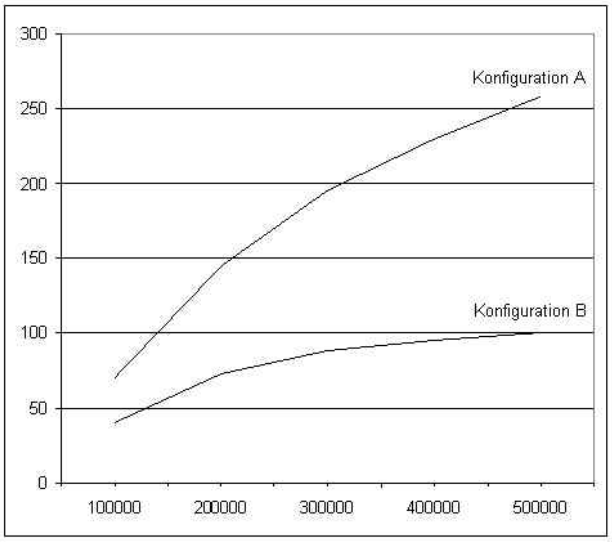

Abbildung 8.19: *Die Ergebnisse des Visibility Cullings gemessen mit zwei Sätzen an Parametern. Die x-Achse beschreibt die Anzahl der Elemente in der Szene, die y-Achse die Zeit pro Frame in Millisekunden.*

Abbildung 8.19 zeigt die Ergebnisse des Occlusion Cullings mit zwei verschiedenen Parametersätzen. Konfiguration A bedeutete dabei $P_1 = 10\%$, $P_2 = 5\%$, $P_3 = 0,1\%$, $P_4 = 1\%$, $P_5 = 500$ und Konfiguration B beinhaltete $P_1 = 5\%$, $P_2 = 3\%$, $P_3 = 0,01\%$, $P_4 = 0,1\%$, $P_5 = 500$. Konfiguration B ist also präziser als Konfiguration A, weil erstere auch AABBs und Polygone mit kleinerer GPM akzeptiert. Obwohl der Aufwand der Sichtbarkeitsbestimmung bei Konfiguration B höher ausfällt als bei Konfiguration A, sind die Laufzeiten im Fall B dennoch niedriger, da der eingesparte Visualisierungsaufwand die Berechnung der Sichtbarkeitsverhältnisse überwiegt. Entscheidend an den Kurven ist, dass der Aufwand mit dem Einfügen der Elemente nicht liniear ansteigt. Die Effizienz des Algorithmus steigt also mit wachsender Zahl der Elemente. Die Abbildungen 8.20, 8.21 und 8.22 zeigen ein paar Beispiele des Verfahrens.

Abbildung 8.20: *Die für die Tests verwendete Stadtszene umfasste bis zu 500000 Gebäude mit durchschnittlich 250 Dreiecken.*

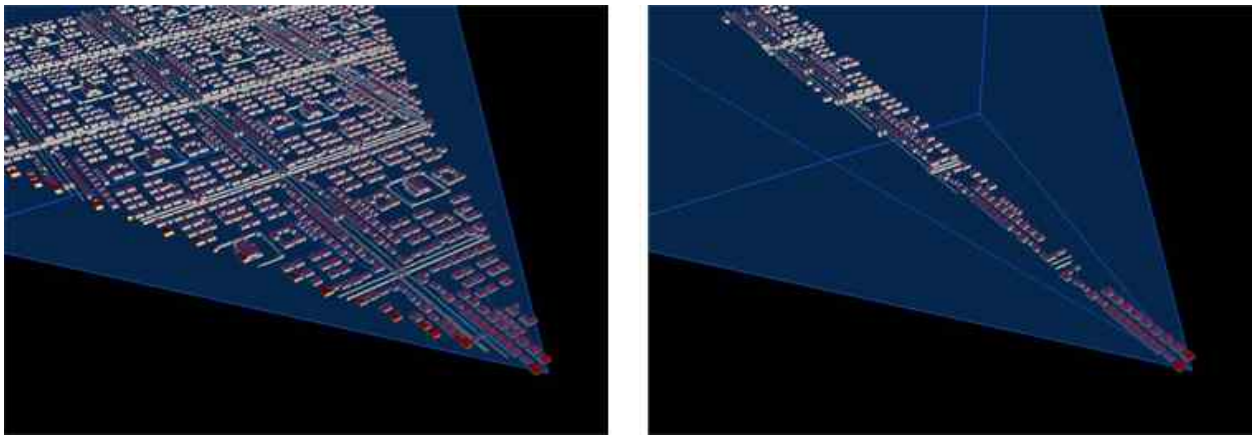

Abbildung 8.21: *Das View Frustum Culling auf der linken Seite identifiziert alle Elemente außerhalb der Sichtpyramide. Das auf der rechten Seite dargestellte Occlusion Culling erkennt dagegen auch Verdeckungen der Elemente untereinander.*

Abbildung 8.22: *Das Occlusion Culling berechnet für jeden Frame eine Menge von Occludern. Die rechte Seite zeigt die selektierten Polygone in gelber Farbe.*

8.6 Zusammenfassung

Dieses Kapitel stellte zunächst zwei Projekte vor, in denen das vorgestellte System zur Anwendung kommt. Danach wurden wichtige Ergebnisse der Arbeit präsentiert.

Bei den beiden Projekten handelt es sich um Huge Ubiquitous Graphical Objects (HUGO) und In3D Viewer. Ersteres ist ein von der Heinz Nixdorf Stiftung gefördertes Forschungsprojekt, letzteres ein kommerzielles Produkt zur Übertragung und Visualisierung geographischer Informationen, welches in Kooperation mit der Firma GIStec GmbH entstand.

Die Ergebnisse der räumlichen Sortierung zeigen, dass der Flaschenhals dynamischer Szenen nicht in der Reorganisation der Baumstrukturen liegt, sondern in der Berechnung der Zielzellen. Ein Grund hierfür ist das Konzept der kompensierenden Bewegungen, das einen deutlichen Geschwindigkeitsgewinn mit sich bringt.

Verfahren zur automatischen Reduktion eines Modelles sind bei sehr einfachen Objekten mit geringer Polygonzahl nur bedingt tauglich. Hier ist oftmals eine Nachbearbeitung von Hand erforderlich. Nichtsdestotrotz liefert das vorgestellte Verfahren visuell ansprechende Ergebnisse bei sehr gut Performanz. Die visuellen Resultate können durch die Berücksichtigung der Zusatzattribute während der Simplifizierung weiter verbessert werden.

Der Scheduler ist in der Lage, die Areas-of-Interest von 400 registrierten Clients in einer Szene mit 500000 Elementen dreimal die Sekunde auszuwerten. Das Konzept der adaptiven Areas-of-Interest reduziert die Anzahl der selektierten Elemente deutlich. Ein Großteil der Rechenleistung wird für die Kodierung der zu übertragenden Elemente verbraucht. Hier ist eine Serverhierarchie mit spezialisierten Nachrichtenservern sinnvoll.

Das vorgestellte Occlusion Culling erlaubt interaktive Frameraten auch in sehr komplexen Szenen. Die Framerate sank bei entsprechender Konfiguration nicht unter 10 Frames pro Sekunde. Die Szenen umfassten dabei 500000 Elementen, was etwa 125 Millionen Dreiecken entsprach.

Kapitel 9

Zusammenfassung und Ausblick

Die in den vergangenen Kapiteln beschriebenen Ideen, Konzepte und Umsetzungen dienen letztlich zwei entscheidenden Zielen: Zum einen der Visualisierung großer, interaktiver und dynamischer 3D Szenen und zum anderen deren adaptiver Verteilung auf heterogene Endgeräte. Entsprechend sind diese beiden Intentionen als grundlegende Anforderungen in Kapitel 2 formuliert. Der Gedanke, sich mit dem Thema der besagten Anforderungen zu beschäftigen, ist nicht einfach aus dem Bauch heraus entstanden. Visuelle Kommunikation ist seit jeher ein bedeutendes Instrument des menschlichen Zusammenlebens. Ausgehend von dem Satz, wonach ein Bild mehr als 1000 Worte ersetzen kann, ist mit den modernen Medien zur Verbreitung und Darstellung von Informationen eine Lawine in Gang gesetzt worden, deren Ende noch nicht abzuschätzen ist. Die Entwicklung entspricht dabei einem logischen Gedankengang: Wenn bereits ein Bild eine große Aussagekraft hat, warum dann nicht viele Bilder? Und warum diese nicht zu einer Bewegung kombinieren? Der Begriff der Animation ist nicht umsonst aus dem lateinischen *animare* entsprungen, was so viel wie beseelen oder beleben bedeutet. Im Gegensatz zu den zweidimensionalen Bildern bieten dreidimensionale Szenarien eine wesentlich größere Vielzahl an Interaktionsmöglichkeiten. Transformationen und Manipulationen der Elemente einer Szene sind nur ein Teil davon. Bei Bedarf lassen sich auch hier zweidimensionale Bilder und Filme mit Hilfe von Kamerafahrten oder Navigation erstellen. Aufgrund der immer weitergehenden Globalisierung der Märkte sowie der Vernetzung der Haushalte liegt der Gedanke nahe, dynamische 3D Grafiken und Szenen zu verbreiten. Hierzu zeigt Kapitel 3, dass im kommerziellen Bereich eine ganze Reihe von Anwendungen existieren, welche aus dem Transfer und der Visualisierung dreidimensionaler Informationen Kapital schlagen. Beispiele sind 3D Produktwerbungen, Virtual Chatrooms und 3D Online Spiele. Allerdings basieren diese Applikationen auf den in Abschnitt 5.1 diskutierten Technologien und unterliegen deshalb den entsprechenden Restriktionen. Insofern reduzieren sich Produktwerbungen auf die Darstellung einzelner Elemente. Virtual Chatrooms nehmen eine bescheidene Grafikqualität zugunsten der Avatare und der Kommunikation ihrer Benutzer in Kauf. 3D Online Spiele verwenden keine Übertragung der visuellen Informationen. Allen gemeinsam ist, dass sie bis auf einige feste Detailstufen keine Adaption an die Leistungsmerkmale der Endgeräte bieten. Das Malheur wird noch an anderer Stelle sichtbar: Bedeutende Firmen schließen sich zu einem Konsortium zusammen, dessen Intentionen sich mit den beiden oben genannten Anforderungen decken. Das Resultat ist mit

VRML ein vielversprechender Ansatz, der jedoch in großen Teilen auf dem bereits existierenden Open Inventor basiert. Leider sind detaillierte VRML Szenen zu groß und unhandlich und müssen außerdem als Ganzes übertragen werden. Anstatt aber nun innovative Wege zu gehen, verweist der aktuelle VRML Nachfolger X3D für die Behandlung der 3D Szenen auf MPEG-4. Die MPEG-4 Spezifikation wiederum basiert im Wesentlichen auf VRML, weshalb die Entwicklung hier ein wenig auf der Stelle tritt.

Wodurch zeichnet sich die vorliegende Arbeit nun aus? Die Idee, 3D Szenen zu übertragen oder zumindest auf verteilten Endgeräten anzuzeigen, ist nicht neu. Funkhouser [Fun96a] bewies bereits im Jahr 1995, dass mit Hilfe von Occlusion Culling Verfahren der Aufwand zur Synchronisation dynamischer, verteilter 3D Szenen reduziert werden kann. Hoppe [Hop96] beschrieb 1996 ein Verfahren zur progressiven Simplifizierung und Verfeinerung von Polygonnetzen. Seine Motivation hierfür war die Anpassung der Modelle an die Leistungsmerkmale eines Endgerätes sowie die schrittweise Übertragung der Informationen aufgrund geringer Bandbreiten. Daneben gibt es mit Teler et al. [TL01] und Hesina et al. [HS98] Veröffentlichungen, die sich mit der Priorisierung und Selektion der zu übertragenden Informationen beschäftigen.

Die Innovation der vorliegenden Arbeit ist die Übertragung großer, dynamischer 3D Szenen auf beliebige, möglicherweise sogar mobile Endgeräte in Echtzeit, wobei selbst der Server einem Standard PC entsprechen darf. Funkhousers Ansatz basiert auf einem PVS Verfahren, weshalb er lediglich in statischen Szenen mit extrem hoher Verdeckungstiefe einsetzbar ist. Zwar sind die Avatare dynamisch, die Welt selbst aber nicht. Zudem berücksichtigt er weder die Übertragung noch die Adaption visueller Informationen. VRML und somit auch MPEG-4 sind in der Lage, dynamische Szenen zu beschreiben, müssen aber vollständig an ein Endgerät übermittelt werden. Hier erfolgt also keinerlei Adaption an die Leistungssmerkmale eines Endgerätes. Um nun auch schwache Endgeräte und Verbindungen mit niedrigen Bandbreiten zu berücksichtigen, ist der Detailgrad üblicher VRML Welten sehr gering. Weder VRML noch MPEG-4 treffen eine Vorgabe, wie große, dynamische Szenen organisiert und selektiv übertragen werden können. Der Peer-to-Peer Ansatz DIVE [CH93a, CH93b, Hag96, FS98] ist aufgrund der Verwendung einer lokalen Kopie nur schwer für die Synchronisation dynamischer Szenen zu gebrauchen. Die Selektion der Daten erfolgt hier eher auf einer semantischen Ebene und schützt nicht vor der Überlastung eines Endgerätes. Ähnlich zu VRML und MEPG-4 fehlt auch im Falle von DIVE eine effiziente räumliche Sortierung dynamischer Szenen. Teler et al. und Hesina et al. gehen nicht auf die Behandlung dynamischer Szenen ein. Obgleich sie die Priorisierung der Elemente einer Szene mittels eines Area-of-Interest Konzepts beschreiben, beziehen sie keine genaue Stellung, wie sie eine Szene repräsentieren und wie sie mit Hilfe dieser Repräsentation die Elemente innerhalb einer oder gar mehrerer Areas-of-Interest identifizieren. Ein derartiges Verfahren für generische, dynamische 3D Szenen ist in Bezug auf mehrere Clients nirgends zu finden. Die Idee, Occlusion Culling Verfahren zu verwenden, erweist sich dabei als zu aufwändig, ohne gleich einen teuren, grafikfähigen Server benutzen zu müssen. Auf dessen Einsatz basiert das VizServer Konzept [SGId] von SGI. Ein millionenschwerer Superrechner rendert lokal alle Frames einer Applikation und sendet diese als Videostream an die Clients. Dieses Verfahren ist in der Praxis einfach umzusetzen, aber für einen Großteil mittelständischer und kleiner Unternehmen nicht erschwinglich. Zudem löst der Ansatz nicht das Problem der Organisation dynamischer 3D

Szenen und bezieht die Clients im Rahmen ihrer Fähigkeiten nicht in die Visualisierung ein. Daher sind zwar auch auf schwachen Endgeräten hochwertige Ausgaben möglich, allerdings erhöht sich die Belastung des Servers mit jedem Client drastisch.

Eine zentrale Rolle im Konzept der vorliegenden Arbeit spielt die in Abschnitt 6.3 beschriebene Repräsentation großer, dynamischer und interaktiver 3D Szenen. Grundlegendes Element einer Szene ist der Animationsagent, der ähnlich dem Konzept der Animationselemente (vergleiche Abschnitt 4.2) nicht nur das visuelle Erscheinungsbild eines dynamischen Bausteins der Szene kapselt, sondern auch dessen spezifisches Verhalten. Der Grund für die Bezeichnung als Agent beruht auf zwei Aspekten: Zum einen werden Agenten in dieser Arbeit lediglich als Erweiterung des komponentenbasierten Programmierparadigmas verstanden. Zum anderen ist es mittels der in Abschnitt 6.3.3 eingeführten Animationslisten möglich, Animationen ausgehend von sehr einfachen Basistransformationen bis auf eine aufgabenorientierte Stufe zu etablieren. Ein Animationsagent entspricht letztlich einer Komponente, die derartige Animationen beziehungsweise Aufgaben ausführen kann. Animationsagenten können zu Animationshierarchien kombiniert werden, wobei die Wurzel der Hierarchie das eigentliche Verhalten der betroffenen Agenten beschreibt. Der Vorteil von Komponenten liegt in der Bereitstellung eines Frameworks, welches die Kommunikationsschnittstellen zwischen den einzelnen Komponenten normiert und somit den transparenten Austausch der Komponenten innerhalb des Frameworks erlaubt. Das in Abschnitt 6.3.5 erläuterte Konzept der Renderer entspricht einem solchen Framework. Bei einem Renderer handelt es sich um eine Komponente, die eine Datenstruktur traversiert, um eine bestimmte Form der Ausgabe zu erzeugen. Renderer kapseln eventuelle betriebssystem- oder hardwarespezifische Abhängigkeiten und ermöglichen zudem die Anpassung der Ausgabe an ein beliebiges Level-of-Abstraction. Sie sind Bestandteil der Plattform und können beim Empfang eines Animationsagenten in selbigen eingesetzt werden, um beispielsweise seine Visualisierung zu ermöglichen.

Das optische Erscheinungsbild eines Animationsagenten wird über einen Elementgraphen beschrieben. Das Prinzip der Elementgraphen ist ähnlich zu gängigen Szenegraphen wie etwa Open Inventor: Während der Traversierung des Graphen wird beim Betreten eines Knotens ein Zustand gesetzt, welcher solange erhalten bleibt, bis er durch den Besuch eines der folgenden Knoten überschrieben wird. Derartige Graphen haben den Vorteil, ein Modell ähnlich einem Baukastenprinzip zusammenfügen zu können. Darüber hinaus sind sie durch das Erzeugen neuer Knotentypen leicht zu erweitern. Im Sinne der effizienten Repräsentation einer Szene beinhalten die Knoten nicht direkt die visuellen Informationen eines Agenten, sondern verweisen auf Einträge in sogenannten Pools. Ein Pool entspricht einem Sammelbecken für die Informationen einer Szene. Zu jedem Datentyp wie etwa geometrischen oder topologischen Daten korrespondiert ein eigener Pool. Ein gesonderter Pool beinhaltet Einträge mit Elementgraphen. Hierdurch ist es möglich, dass mehrere Animationsagenten sich den gleichen Elementgraphen teilen und wiederum verschiedene Elementgraphen identische Pooleinträge referenzieren. Auf diese Weise werden Redundanzen bei der Repräsentation einer Szene vermieden. Neben den Pools zur Konservierung der visuellen Informationen einer Szene existiert mit dem Animation Pool ein Container, der vordefinierte Animationen aufnimmt. Die Pooleinträge kapseln nahezu die gesamten Informationen einer Szene in einer speichereffizienten Form. Insofern ist ihre Übertragung der Grundstein für eine erfolgreiche Umsetzung der beiden genannten Anforderungen.

Techniken wie etwa das Occlusion Culling Verfahren von Bartz et al. [BMH99] werden häufig als für dynamische Szenen ungeeignet erachtet, weil sie eine räumliche Sortierung der Elemente in Form einer Bounding Volume Hierarchie oder eines Raumunterteilungsbaumes benötigen. Grund ist die Streitfrage, ob das Erstellen einer derartigen Sortierung als aufwändige Vorberechnung zu zählen ist oder in Echtzeit dynamischen Vorgängen angepasst werden kann. Die vorliegende Arbeit geht von letzterer Ansicht aus und beschreibt eine Lösung des Problems in Abschnitt 6.3.4. Die Intention einer räumlichen Sortierung der Elemente beruht auf der Identifikation entsprechender Kohärenzen. Hierüber sind verallgemeinernde Aussagen möglich, die von Applikationen wie Kollisionserkennungen oder Occlusion Culling Verfahren ausgewertet werden. Ist beispielsweise eine Zelle eines Raumunterteilungsbaumes in Bezug auf einen Betrachter unsichtbar, so sind auch alle Elemente innerhalb der Zelle unsichtbar. Bounding Volume Hierarchien sind für dynamische Szenen nur bedingt einsetzbar. Ihr Problem ist das schnelle Finden benachbarter Elemente, um eine massive Überlappung der einzelnen Zellen zu vermeiden. In nicht seltenen Fällen ist aufgrund einer Verletzung der Kapazität δ einer Zelle die komplette Reorganisation der Hierarchie nötig. Raumunterteilungsbäume wiederum haben das Problem der Schnittelemente, die sich nicht eindeutig einer Zelle zuordnen lassen. Glücklicherweise existiert mit den Nine-Area-Trees von Chang et al. [CLC03] ein Lösungsansatz für die Behandlung der Schnittelemente im zweidimensionalen Fall.

Aus diesem Grund beschreibt Abschnitt 6.3.4 ein neuartiges Konzept für einen Raumunterteilungsbaum, der mit einer Erweiterung des Ansatzes von Chang et al. in den dreidimensionalen Raum aufwartet. Mit diesem Baum sind drei dynamische Vorgänge definiert: Das Hinzufügen eines Animationsagenten, das Entfernen eines Agenten sowie das Transformieren eines Agenten innerhalb der Szene. Jeder Vorgang für sich betrachtet kann eine Verletzung der Kapazität δ einer Zelle verursachen, welche gleichzeitig die Reorganisation des Baumes impliziert. Im Falle einer derartigen Reorganisation sind zwei Operationen definiert, nämlich die Divide und die Unite Operation. Während erstere eine Zelle beim Überschreiten der Kapazität δ in mehrere Tochterzellen unterteilt, vereint letztere einen kompletten Teilbaum beim Unterschreiten der Kapazität wieder zu einer Zelle. Grundlegendes Ziel des Konzepts ist aber, sowohl die Häufigkeit der Reorganisationen als auch die Anzahl der Divide beziehungsweise Unite Operationen zu verringern. Hierzu wird in Bezug auf die beiden Operationen ein Toleranzbereich eingeführt, weshalb jede Zelle über eine Kapazität δ_{min} als auch eine Kapazität δ_{max} verfügt. Damit ist eine Unite Operation nicht beim sofortigen Unterschreiten der Kapazität δ beziehungsweise nun δ_{max} erforderlich, sondern erst mit dem Unterbieten von δ_{min}. Um die mehrfache Verletzung der Kapazitäten durch die Bewegung eines einzelnen komplexen Elements zu vermeiden, werden die Agenten einer Animationshierarchie in einem Hierarchieknoten gekapselt. Die Idee der Hierarchieknoten basiert auf der Beobachtung, dass eine Animationshierarchie üblicherweise räumlich benachbarte Agenten beinhaltet, welche sich tendenziell in eine äquivalente Richtung bewegen. Weiterhin ist die Einsortierung der Animationsagenten anhand einer Temporary Bounding Box [SG96] möglich. Diese definiert hinsichtlich eines Agenten einen Raum und ein Zeitintervall, währenddessen der Agent den Raum nicht verlässt. Innerhalb des Intervalls kann ein Animationsagent keine Verletzung der Kapazität einer Zelle verursachen. Das Konzept der sich gegenseitig kompensierenden Bewegungen sammelt in der Zeitspanne zwischen zwei Simulationszeitschritten alle dynamischen Vorgänge und betrachtet bei der nunmehr lediglich pro Zeitschritt anfallenden Reorganisati-

on ausschließlich den Nettowert der Bewegungen. Zudem bietet das Konzept die Möglichkeit paralleler Lese- und Schreibzugriffe auf die Zellknoten des Raumunterteilungsbaumes. Da selbiger von einer ganzen Reihe anderer Applikationen frequentiert wird, bedeutet dies einen enormen Vorteil.

Nach der erfolgreichen Repräsentation großer, dynamischer und interaktiver 3D Szenen ist der nächste Schritt die Übertragung einer derartigen Szene an mehrere Endgeräte. Dabei sind sowohl die Leistungsmerkmale der betreffenden Geräte als auch die Bandbreite der Verbindung zwischen Server und Client zu berücksichtigen. Gemäß Kapitel 2 entsprechen die Leistungsmerkmale eines Endgerätes seiner Rechenleistung, seinem Speichervermögen, seiner Grafikfähigkeit sowie seinen Interaktionsmöglichkeiten. Diese vier Eigenschaften werden durch einen normierten Benchmarkindex repräsentiert, dessen Kalkulation automatisch durch die Zeitmessung wichtiger Basisoperationen erfolgt. Der Benutzer hat jedoch die Option, einen eigenen Index vorzugeben. In Kombination mit einem ebenfalls normierten, dafür jedoch dynamischen Index hinsichtlich der Bandbreite ergibt sich ein prozentualer Faktor, welcher die maximale Menge der an einen Client zu übertragenden Daten spezifiziert. Invasive Verfahren erreichen eine Adaption der Daten an die Leistungsmerkmale durch die Manipulation beziehungsweise Vereinfachung der Daten selbst. Demgegenüber reduzieren nicht-invasive Vorgehensweisen den Aufwand mittels einer geschickten Selektion der Daten.

Ein Beispiel für invasive Verfahren sind Technologien zur automatischen Level-of-Detail Generierung, die sich nach Abschnitt 4.3 unter anderem in Verfahren zur Entfernung von Geometrie, in Resampling Ansätze und in Verfeinerungsalgorithmen klassifizieren lassen. Die ursprüngliche Idee der Level-of-Detail [Cla76] beruht darauf, Elemente mit geringem Einfluss auf den Sichtbereich eines Betrachters in einem niedrigen Detailgrad darzustellen und so den Aufwand ihrer Visualisierung einzuschränken. Insbesondere mit der progressiven Simplifizierung von Hoppe [Hop96] entstand dann die Absicht, nahezu stufenlose Level-of-Detail zu generieren und diese für die exakte Anpassung an verschiedene Leistungsmerkmale zu nutzen. Als Innovation bietet der Ansatz von Hoppe die schrittweise Übertragung und Verfeinerung eines Modells mit Hilfe mehrerer Datenpakete, weshalb die Beschreibung eines Elements nicht mehr als Ganzes transferiert werden muss. Unglücklicherweise haben die in Abschnitt 4.3 erläuterten Techniken eine Reihe von Nachteilen. Modelle, die über implizite Funktionen, parametrische Funktionen, Multiresolution Analysis oder Unterteilungsflächen repräsentiert sind, erfordern vor ihrer Visualisierung eine aufwändige Transformation in Dreiecksnetze, da nur diese von der Grafik-Hardware eines Endgerätes unterstützt werden. Ein Verwenden der Grafik-Hardware ist aber für eine Echtzeitausgabe unabdingbar. Resampling Techniken wie etwa von Rossignac et al. [RB93] bieten keine Kriterien, um das Ergebnis des Verfahrens abzuschätzen. Im Besonderen ist die Vorhersage der verbleibenden Dreiecke einer Detailstufe nicht möglich, was jedoch für die exakte Adaption an die Leistungsmerkmale erforderlich ist. Die Ansätze zur progressiven Simplifizierung von Polygonnetzen nach Hoppe [Hop96] und Popovic et al. [PH97] lösen ein Optimierungsproblem unter Berücksichtigung von Energiefunktionen. Beide Verfahren erreichen somit für jedes vereinfachte Modell eine sehr gute Approximation an das Originalmodell, müssen aber in Bezug auf das vereinfachte Modell neue Scheitelpunkte berechnen und einfügen. Aus diesem Grund sind sowohl die Simplifizierung auf Seiten des Servers als auch die Verfeinerung auf Seite des Clients rechenintensive Vorgänge. Der Ansatz von Schröder [Sch97] ermöglicht über die Auflösung

der Topologie eines Modells eine höhere Kompression, welches in einem Basismesh mit einer sehr geringen Zahl an Dreiecken resultiert. Schröder macht aber keine näheren Angaben zu eventuellen Zusatzattributen sowie zu seinem progressiven Format.

Das in Abschnitt 6.4 erläuterte Verfahren entspricht ebenfalls einem progressiven Simplifizierungsverfahren, wartet allerdings mit mehreren Innovationen auf. Die neue Fehlermetrik entstammt keinem mathematisch eleganten Konzept wie etwa bei Hoppe [Hop96]. Auch ist im Gegensatz zu Klein et al. [KLS96] die Einhaltung von Fehlerschranken nicht gegeben. Dafür ermöglicht die Metrik aber eine sehr schnelle Berechnung des Fehlers eines Scheitelpunktes bei gleichzeitig ansprechenden visuellen Ergebnissen. Die zur Rekonstruktion einer Kantenkollabierung anfallenden progressiven Daten beanspruchen nur einen sehr geringen Speicherbedarf und erlauben zudem die extrem einfache und kostengünstige Verfeinerung auf einem Client. Die hierzu verwendeten Datenstrukturen bieten eine konsistente Repräsentation des Modells, welche zum Zwecke der Visualisierung ohne jegliches Konvertieren oder Traversieren an eine Grafik-Bibliothek wie OpenGL übergeben werden kann. Auf diese Weise erhält der Benutzer augenblicklich mit jedem neu eingefügten Dreieck ein optisches Feedback und nicht erst dann, wenn eine bestimmte Detailstufe komplett empfangen wurde. Zusatzattribute werden pro Scheitelpunkt als auch pro Fläche unterstützt und liegen auch nach der Simplifizierung noch als getrennte Datenströme vor. Hierdurch ist die spezifische Kodierung und Dekodierung der Datenströme durch geeignete Codecs gegeben. Scheitelpunkte wie auch Zusatzattribute beschreiben innerhalb der Datenströme eine zu den Kantenkollabierungen inverse Sequenz, wodurch eine Aufteilung der Ströme in einzelne Datenpakete möglich ist. Die Pakete können dann clientseitig einfach aneinander gereiht werden. Ähnliches gilt auch in Bezug auf die topologischen Informationen, wobei hier aber eine gesonderte Behandlung im Sinne der progressiven Daten erforderlich ist. Die Implementierung des Verfahrens als Elementgraphrenderer erlaubt die Verarbeitung beliebiger nach dem Konzept der Elementgraphen erstellter Modelle. Im Anschluss an die Simplifizierung identifiziert ein Elementgraph neben dem Basismesh auch die für dessen Verfeinerung notwendigen Daten.

Als nicht-invasive Verfahren werden häufig die in Abschnitt 4.6 eingeführten Occlusion Culling Verfahren bemüht. Deren Intention liegt in der Identifikation der hinsichtlich eines Betrachters sichtbaren Elemente einer Szene. Im Gegensatz zu anderen Vorgehensweisen gelten Occlusion Culling Ansätze als ausgabesensitiv, weil ihre Laufzeit lediglich proportional zur Anzahl der sichtbaren Elemente ausfällt. Dabei ist das Anwendungsgebiet der Occlusion Culling Verfahren nicht ausschließlich auf die Visualisierung beschränkt. Vielmehr kommen sie in Kombination mit einem Area-of-Interest Konzept auch für die Selektion der zu übertragenden Elemente einer Szene in Frage. In diesem Sinne repräsentieren Occlusion Culling Verfahren ein visuelles Kriterium für die Auswahl der Elemente. Konsequenterweise werden nur diejenigen Elemente an einen Client übertragen, die sich dort im Sichtbereich des Benutzers befinden. Nach Funkhouser [Fun96a] ist das gleiche Prinzip ebenfalls für die Synchronisation dynamischer Szenen einsetzbar. Allerdings sind Occlusion Culling Verfahren zumindest in Bezug auf generische, dynamische Szenen derart aufwändig, dass selbst unter Zuhilfenahme der Grafik-Hardware die Echtzeitvisualisierung einer solchen Szene zur Herausforderung wird. Da dies schon für die Evaluation einer einzelnen Sichtpyramide zutrifft, sind solche präzisen Vorgehensweisen auf einem Server, der entsprechend der Zahl der registrierten Clients bedeutend mehr Areas-of-Interest berücksichtigen muss, nicht praktikabel.

Hesina et al. [HS98] verwenden daher anstelle der Sichtpyramide eine wesentlich einfachere, kreisförmige Area-of-Interest. Das Kriterium wird an dieser Stelle von der exakten Sichtbarkeitsbestimmung auf die Kalkulation des Abstandes eines Elements zum Betrachter reduziert. Wie schon zuvor angedeutet, gehen Hesina et al. aber ähnlich zu Teler et al. [TL01] nicht auf die eigentliche Identifizierung der Elemente innerhalb der Areas-of-Interest ein.

Demgegenüber beschreiben die Abschnitte 6.5 und 6.7.1 nicht nur ein neuartiges Area-of-Interest Konzept, sondern auch dessen effiziente Auswertung. Eine Area-of-Interest wird dabei durch drei ineinander verschachtelte achsenparallele Zonen beschrieben, die jeweils in Bezug auf die Übertragung eine bestimmte Priorität repräsentieren. Die innerste Zone umschließt die Sichtpyramide des Betrachter und bedeutet für enthaltene Animationsagenten die höchste Priorität. Die mittlere Zone ist ausschließlich bei der Transformation der Sichtpyramide von Bedeutung und beschreibt die zweithöchste Priorität. Die äußerste Zone dient der präventiven Übertragung von Animationsagenten, die in naher Zukunft in den Sichtbereich des Betrachters gelangen könnten. Sie repräsentiert die niedrigste Priorität. Um die Verschwendungsrate, d.h. die Anzahl unnötigerweise übertragener Animationsagenten zu reduzieren, erfolgt eine Anpassung der drei Zonen an die zu erwartende Navigation des Betrachters. Entsprechend den Ergebnissen eines Feldtests setzt sich eine Navigation aus drei grundlegenden Situationen zusammen. Dabei handelt es sich um positionsbezogene Aktionen, geradlinige Bewegungen sowie bogen- beziehungsweise kurvenförmige Bewegungen. Ähnlich dem Vorschlag von Teler et al. geht der Ansatz von einer kontinuierlichen Fortsetzung einer einmal gestarteten Form der Navigation aus.

Gemäß ihrer Priorität werden die Zonen aller Areas-of-Interest in jeweils drei Clientbäume eingefügt, welche ihrerseits ausschließlich eine Dringlichkeitsstufe repräsentieren. Die Intention der Clientbäume ist die Minimierung der Vergleiche zwischen Zonen und Animationsagenten, indem sie nicht nur die räumlichen Kohärenzen der Agenten, sondern auch die der Zonen berücksichtigen. Im Anschluss an den Aufbau eines Clientbaumes wird die Szene in Form des Raumunterteilungsbaumes gegen den Clientbaum getestet. Das Resultat des Tests ist nicht nur die Aussage, ob sich ein Animationsagent innerhalb einer Area-of-Interest befindet, sondern die genaue Angabe seiner Priorität in Bezug auf alle Clients. Die Idee, pro Dringlichkeitsstufe einen eigenen Clientbaum zu erstellen, beruht auf zwei Aspekten. Zum einen beschreiben die Animationsagenten mit hoher Priorität nur eine kleine Teilmenge der Agenten mit niedrigerer Prioriät. Zum anderen müssen erstere öfter überprüft werden als letztere, da sich Änderungen im Falle der als wichtig eingestuften Agenten direkt auf den Sichtbereich des Betrachters auswirken können. Diese Kriterien finden im Konzept der Zyklen ihre Berücksichtigung.

Ein entscheidender Vorteil des Ansatzes liegt darin, dass pro Test gegen einen Clientbaum ein Knoten des Raumunterteilungsbaumes höchstens einmal betreten wird. Das ist äußerst günstig für eine Out-of-Core Strategie, weil hier jeder Zugriff auf einen Knoten teure Aus- und Einlagerungsoperationen nach sich ziehen kann. Die durch den Test ermittelten Prioritäten bilden die Grundlage für eine derartige Strategie auf Seite des Servers. Demnach sind Animationsagenten innerhalb der kleinsten Zone einer Area-of-Interest so lange wie möglich im Hauptspeicher zu halten. Agenten außerhalb jeglicher Areas-of-Interest sind dagegen geeignete Kandidaten für eine Auslagerung. In Kombination mit einem Last Recently Used Ansatz ist eine automatische Out-of-Core Behandlung gegeben.

Durch die Identifizierung der Animationsagenten innerhalb der Areas-of-Interest ist es nun möglich, die Agenten gezielt an einen Client zu übertragen. Abschnitt 6.6 beschreibt dazu auf Seite des Servers das Erstellen eines Abhängigkeitsgraphen, in den sowohl die Agenten als auch ihre Elementgraphen und Pooleinträge eingepflegt werden. Dieser Abhängigkeitsgraph definiert zusammen mit den ermittelten Prioritäten der Animationsagenten eine Übertragungssequenz seiner Knoten. Mit dem Empfang der Daten erstellt der Client einen entgegengesetzten Abhängigkeitsgraphen und kann so die Vollständigkeit der Informationen kontrollieren. Weiterhin ermöglicht der Graph dort die Synchronisation der durch das progressive Format getrennten Datenströme. Sobald ein Knoten des Abhängigkeitsgraphen und somit auch ein Animationsagent in einem für die Visualisierung geeigneten Zustand ist, wird er in die Szenenbeschreibung des Clients eingetragen.

Mit der Visualisierung der Animationsagenten durch den Client schließt sich der Kreis. Obgleich ein Client in der Regel nur eine Teilmenge des Servers beinhaltet, ist es dennoch nicht sinnvoll, die gesamte Szene der Grafik-Hardware zur Bearbeitung zu überlassen. Stattdessen erlaubt das Konzept der Renderer den Einsatz verschiedener Techniken für die Reduktion des Aufwands. Das in Abschnitt 6.7.2 erläuterte Occlusion Culling Verfahren ist ein Beispiel einer derartigen Technik. Zwar ist der Ansatz von Bartz et al. [BMH99] aufgrund seiner Berücksichtigung der neuen Grafik-Chips nicht zu vernachlässigen, allerdings gehören das HP-Flag sowie die NVidia-Occlusion-Query noch nicht zum Standard aktueller Grafik-Bibliotheken. Der Vorteil des in Abschnitt 6.7.2 beschriebenen Verfahrens liegt neben seiner Effizienz in der ausschließlichen Verwendung des OpenGL Standards sowie in der einfachen Parametrisierung seiner Präzision. Letzteres ermöglicht gemäß der in Kapitel 2 verlangten Adaptivität der Algorithmen eine Anpassung des Rechenaufwands an die Leistungsmerkmale eines Endgerätes.

Um eine Adaption der Interaktionsmöglichkeiten zu garantieren, beschreibt Abschnitt 6.7.3 eine Navigationsschnittstelle, die sowohl für Standard PCs als auch für PDAs geeignet ist. Die Intention hierbei beruht darauf, dem Benutzer die Gelegenheit zu geben, auf dem PC gewonnene Erfahrungen auch auf dem PDA einzusetzen. Die erläuterte Navigationsschnittstelle ist im Wesentlichen der Umsetzung bekannter Spiele wie beispielsweise Ultima Underworld entnommen.

In dem beschriebenen Ansatz zur kollaborativen Visualisierung großer, dynamischer und interaktiver 3D Szenen auf verteilten Endgeräten steckt durchaus noch Potential für Verbesserungen und Erweiterungen. Funkhouser gibt beispielsweise den Anstoß, das Konzept des Servers auf eine Hierarchie von Servern zu erweitern. Dabei sind mehrere Stufen denkbar. Ein großer Teil der Rechenkapazität geht mit dem Kodieren und Versenden der Nachrichten verloren. In einer ersten einfachen Version könnte somit eine Gruppe weiterer Server mit dem Versenden der Nachrichten betraut werden. Diese Vorgehensweise macht hauptsächlich bei Multicasts Sinn, wie sie bei der Synchronisation dynamischer Szenen zur Anwendung kommen. In einer zweiten Version könnte auch die Kompression und Kodierung von anderen Servern übernommen werden. Voraussetzung ist hier eine sehr schnelle Verbindung zwischen den Servern mit hoher Bandbreite. Der Hauptserver sendet die Informationen unkodiert an seine Partner, welche die Nachrichten kodieren und entweder selbst an die Clients versenden oder die Aufgabe wieder anderen Servern belassen.

Eine Hierarchisierung der Server in einer derartigen Form lässt sich noch sehr einfach realisieren, da im Grunde genommen keine Änderungen an der Architektur vorzunehmen sind. Anders sieht das bei einer vollständigen Portierung des Servers in einen PC Cluster aus. Hier ist eine grundsätzliche Entscheidung zu treffen: Soll jeder Knoten des Clusters die gleichen Fähigkeiten und Aufgaben erhalten oder soll jeder Knoten einen ganz spezifischen Part übernehmen? Beide Lösungen haben sicherlich ihren Reiz. Im Falle der ersten Lösung ist durch die Redundanz der Knoten beispielsweise eine gewisse Ausfallsicherheit gegeben. Weiterhin bestehen dort gute Möglichkeiten eines effizienten Load Balancing. Die zweite Lösung erlaubt dagegen durch die Spezialisierung der Knoten eine bessere Ausnutzung der Fähigkeiten eines Gerätes. Beispielsweise können nur einige Knoten über eine Grafik-Hardware verfügen.

Wie auch immer die Entscheidung ausfällt, es bleibt ein schwerwiegendes Problem zu lösen. In fast allen Berechnungen des Servers sind die Daten der Szene involviert. Wie können nun diese Daten den Knoten des Clusters zugänglich gemacht werden? Dabei ist zu berücksichtigen, dass mit einer großen Szene enorme Datenmengen anfallen können. Es wäre zum Beispiel sehr aufwändig, vor der Selektion der zu übertragenden Animationsagenten erst die gesamte räumliche Sortierung auf einen Knoten übertragen zu müssen. Im Falle der ersten der beiden genannten Lösungsvorschläge wäre es denkbar, dass jeder Knoten einen räumlichen Bereich der Szene erhält und diese eigenständig verwaltet. Durch das äquivalente Leistungsvermögen der Knoten kann jeder die anfallenden Aufgaben bewältigen. Allerdings ist die Skalierbarkeit eines derartigen Ansatzes von vornherein auf die Anzahl der Knoten beschränkt. Hinsichtlich der zweiten Lösung muss genau bedacht werden, welche Daten der jeweilige Knoten für seine spezifische Aufgabe benötigt.

Abbildung 9.1 illustriert eine mögliche Erweiterung der in Abschnitt 6.4 erläuterten progressiven Simplifizierung eines Modells. Zwar berücksichtigt das Verfahren eventuelle Zusatzattribute in Form der resultierenden Datenströme, es berechnet aber in Bezug auf Texturkoordinaten keine neuen Informationen. Dies kann zu unerwünschten Ergebnissen führen. Die erste Reihe in Abbildung 9.1 zeigt ein Gesicht, welches aus einem Polygonnetz mit 13000 Dreiecken besteht. Über das Polygonnetz ist eine Textur definiert. Die zweite Reihe zeigt nun das gleiche Gesicht nach der Simplifizierung auf 100 Dreicke mittels eines im Projekt Ubiquitous Graphical Objects implementierten QEM Verfahrens [GH97]. Diese Version des Ansatzes von Garland et al. berücksichtigt keine Texturkoordinaten, weshalb das Gesicht einen deformierten Eindruck erweckt. Reihe drei illustriert das auf 100 Dreiecke simplifizierte Gesicht nach der Anwendung des erweiterten QEM Verfahrens [GH98] von Garland et al. Obgleich Reihe zwei und drei die gleiche Zahl an Dreiecken aufweisen, ist das visuelle Ergebnis der letzteren deutlich besser und nur schwer vom Original zu unterscheiden.

Leider ist die Berechnung der Texturkoordinaten nicht trivial und mit einem entsprechenden Rechenaufwand verbunden. Dies gilt insbesondere für Modelle mit einem höheren Genus. Im Falle einer geeigneten Lösung können die ermittelten Werte in die progressiven Daten integriert werden, welche in Abbildung 6.21 dargestellt sind.

Ein weiteres Problem betrifft ebenfalls die Texturen. Progressive Texture Konzepte beruhen analog dem DjVu Format zumeist auf einer Wavelet Transformation. Damit sind derartige Formate in Bezug auf den Bildraum einer Textur nicht im eigentlichen Sinne progressiv. Stattdessen wird zunächst ein Bild mittels der Wavelet Transformation in einen Bitstream

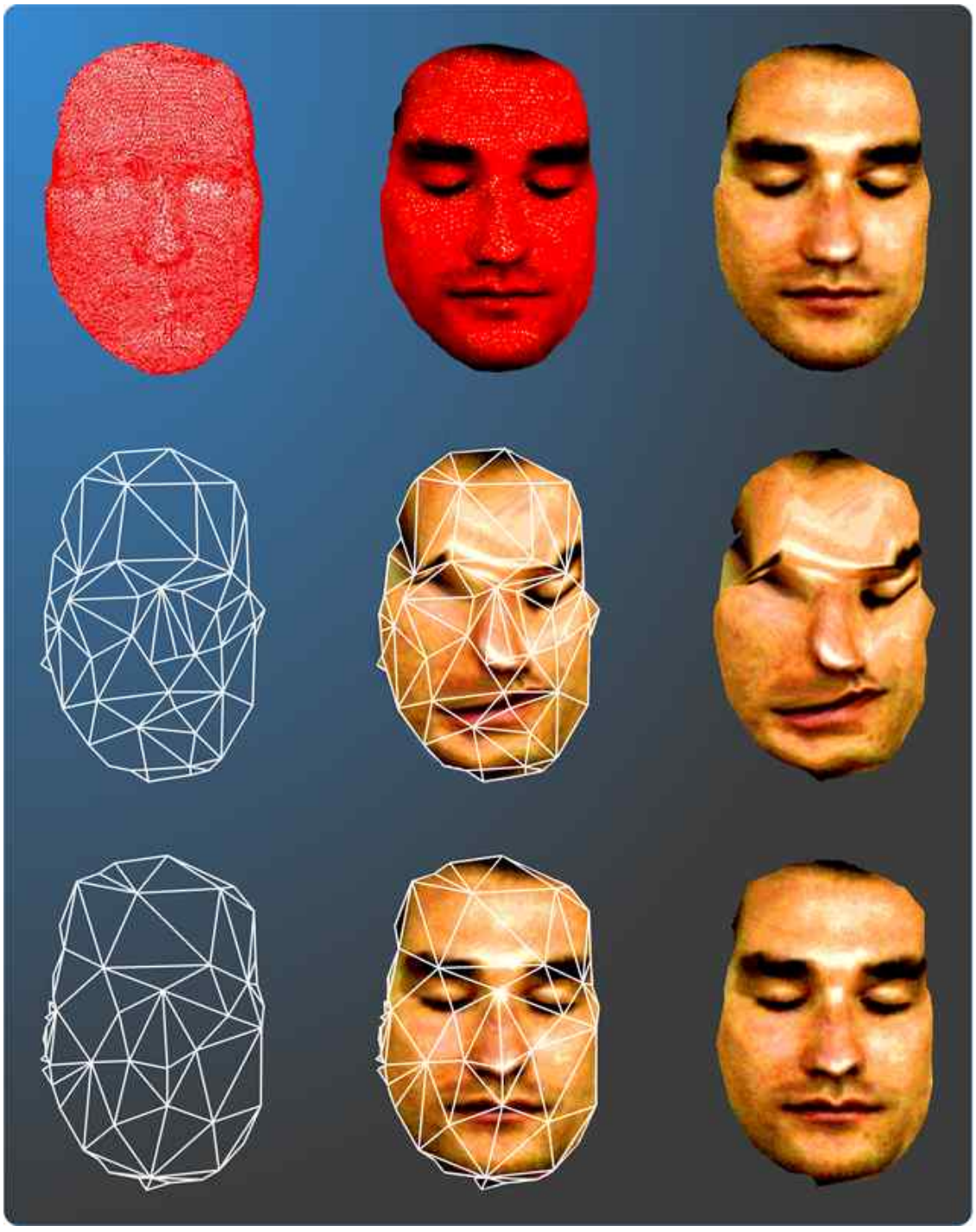

Abbildung 9.1: *Die oberste Reihe zeigt das Original mit etwa 13000 Dreiecken. Die beiden unteren Reihen stellen das jeweils auf 100 Dreiecke reduzierte Modell dar. Werden die Texturkoordinaten während der Simplifizierung nicht neu berechnet so kann es zu den in der zweiten Reihe zu sehenden Deformierungen kommen. Die dritte Reihe zeigt das Ergebnis bei Berücksichtigung der Texturkoordinaten mittels eines erweiterten QEM Verfahrens [GH98].*

kodiert. Dieser Bitstream verfügt über eine hohe Kompression und kann paketweise übertragen werden. Der Bitstream ist allerdings in seiner Form nicht zu visualisieren. Vielmehr muss

wiederum eine Wavelet Transformation zum Einsatz kommen, um das Bild zu rekonstruieren. Mit jedem neuen Datenpaket ist eine weitere Anwendung der Transformation erforderlich, wobei Wavelet Transformationen insbesondere bei Bildern mit höheren Auflösungen äußerst teuer ausfallen. An dieser Stelle wäre entweder eine vollständig neue Entwicklung oder zumindest das Experimentieren mit dem von aktuellen Grafikchips unterstützten S3TC Formats wünschenswert.

Literaturverzeichnis

[3DS] 3d studio max. http://www.3dmax.com.

[ACT] Active worlds. http://www.activeworlds.com.

[ACT00] S. Ar, B. Chazelle, and A. Tal. Self-customized bsp-trees for collision detection. *Computational Geometry - Theory and Applications*, pages 23–29, 2000.

[AF99] B. Aronov and S. Fortune. Approximating minimum weight triangulations in three dimensions. *Discrete Computer Geometry*, 21(4):527–549, March 1999.

[Age] Inc. AgentSheets. Agentsheets. http://www.agentsheets.com.

[AM00] U. Assarsson and T. Moller. Optimized view frustum culling algorithms for bounding boxes. *Journal of Graphics Tools*, 5(1), 2000.

[APB87] B. Arnaldi, T. Priol, and K. Bouatough. A new space subdivision method for raytracing csg modelled scenes. *The visual computer*, 3:98–108, 1987.

[App68] A. Appel. Some techniques for shading machine renderings of solids. In *AIFPS 1968 Spring Joint Computer Conference*, pages 37–45, 1968.

[AS97] L.S. Avila and W. Schroeder. Interactive visualization of aircraft and power generation engines. *IEEE Visualization*, pages 483–486, 1997.

[Ass] Robert McNeel & Associates. Rhinoceros. http://www.rhino3d.com.

[B$\acute{7}$2] P. Bézier. *Numerical Control, Mathematics and Applications*. John Wiley & Sons, 1972.

[Bau75] B.G. Baumgart. A polyhedron representation for computer vision. In *National Computer Conference Proceedings*, pages 589–596. AFIPS, 1975.

[BC93] W. Bricken and G. Coco. The veos project. Technical report, Human Interface Technology Laboratory, University of Washington, 1993.

[Ben75] J.L. Bentley. Multidimensional binary search trees used for associative search. *Communications of the ACM*, 18(9):509–517, September 1975.

[Ben79] J.L. Bentley. Data structures for range searching. *ACM Computing Surveys*, 11(4):397–409, December 1979.

[BGH+90] C. Blanchard, S. Gurgess, Y. Harvill, J. Lanier, A. Lasko, M. Oberman, and M. Teitel. Reality build for two: A virtual reality tool. *ACM SIGGRAPH Special Issue on 1990 Symposium on Interactive 3D Graphics*, pages 35–36, 1990.

[BGRP01] S. Benford, C. Greenhalgh, T. Rodden, and J. Pycock. Collaborative virtual environments. *Communications of the ACM*, 44(7), 2001.

[BHML92] B. Blau, C.E. Hughes, M.J. Moshell, and C. Lisle. Networked virtual environments. *ACM SIGGRAPH Special Issue on 1992 Symposium on Interactive 3D Graphics*, pages 157–164, 1992.

[BHS98] J. Bittner, V. Havran, and P. Slavik. Hierarchical visibility culling with occlusion trees. In *Proceedings of Computer Graphics International 1998*, pages 207–219, 1998.

[BMH98] D. Bartz, M. Meissner, and T. Huettner. Extending graphics hardware for occlusion queries in opengl. In *Proceedings Workshop on Graphics Hardware '98*, pages 97–104, 1998.

[BMH99] D. Bartz, M. Meissner, and T. Huettner. Opengl-assisted occlusion culling for large polygonal models. *Computer & Graphics*, 23(5):667–679, 1999.

[Bou70] W.J. Bouknight. A procedure for generation of three-dimensional half-toned computer graphics representations. *Communications of the ACM*, 13(9):527–536, September 1970.

[Bro97] W. Broll. Distributed virtual reality for everyone - a framework for networked vr on the internet. *IEEE Virtual Reality Annual International Symposium 1997 (VRAIS '97)*, 1997.

[BS70] I.S. Berisin and N.P. Shidkow. *Numerische Methoden I.* VEB-Verlag der Wissenschaft, 1970.

[BS01] E. Blechschmitt and J. Sahm. An agent-based system's architecture to describe workflows in the office. *e-Business and e-Work Conference (e2001)*, October 2001.

[BSS+01] D. Bartz, D. Staneker, W. Strasser, B. Cripe, T. Gaskins, K. Orton, M. Carter, Johannsen A., and J. Trom. Jupiter: a toolkit for interactive large model visualization. In *Proceedings of the IEEE 2001 Symposium on Parallel and Large-Data Visualization and Graphics*, pages 129–134, 2001.

[BSS03] H. Birthelmer, I. Soetebier, and J. Sahm. Efficient representation of triangle meshes for simultanous modification and rendering. In *Proceedings of International Conference on Computational Science Graphics 2003 (ICCS 2003), Part I*, pages 925–934, 2003.

[Cat74] E. Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. PhD thesis, Computer Science Department, University of Utah, Salt Lake City, UT, 1974.

[CC78] E. Catmull and J. Clark. B-spline surfaces on arbitrary topological meshes. *Computer-Aided Design*, 10(6):350–355, 1978.

[Cd59] P. Casteljau de. *Outillage méthodes calcul*. André Citroen Automobiles S.A., 1959.

[CD99] S.W. Cheng and T.K. Dey. Approximate minimum weight steiner triangulation in three dimension. In *Proceedings of the 10th annual ACM-Symposium on Discrete Algorithms*, pages 205–214, 1999.

[CDG+93] J. Calvin, A. Dickens, B. Gaines, P. Metzger, D. Miller, and D. Owen. The simnet virtual world architecture. In *Proceedings of IEEE Virtual Reality Annual International Symposium*, pages 450–455, September 1993.

[CET99] T.K. Capin, J. Esmerado, and D. Thalmann. Dead-reckoning algorithms for streaming virtual human data. *IEEE Transactions on Circuits and Systems for Video Technology*, 1999.

[CF89] N. Chin and S. Feiner. Near real-time shadow generation using bsp trees. In *Proceedings of SIGGRAPH 1989*, pages 99–106. ACM SIGGRAPH, 1989.

[CGL+98] J.H.P Chim, M. Green, R.W.H. Lau, H.V. Leong, and A. Si. On caching and prefetching of virtual objects in distributed virtual environments. *ACM Multimedia*, pages 483–486, 1998.

[CH93a] C. Carlsson and O. Hagsand. Dive - a multi user virtual reality system. *IEEE VRAIS '93*, September 1993.

[CH93b] C. Carlsson and O. Hagsand. Dive - a platform for multi-user virtual environments. *Computer & Graphics*, 17(6), 1993.

[Cha74] G.M. Chaikin. An algorithm for high speed curve generation. *Computer Vision, Graphics and Image Processing*, 3(4):346–349, 1974.

[CJE+98] T.K. Capin, M. Jovovic, J. Esmerado, A. Aubel, and D. Thalmann. Efficient network transmission of virtual human bodies. *IEEE Computer Animation '98*, pages 41–48, 1998.

[CKS02] W. Correa, J. Klosowski, and C. Silva. Interactive out-of-core rendering of large models. Technical Report TR-653-02, Princeton University, 2002.

[Cla76] J. Clark. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, 19(10):547–554, 1976.

[CLC03] Y. Chang, C. Liao, and H. Chen. Na-trees: A dynamic index for spatial data. *Journal of Information Science and Engineering*, 19(1):103–139, January 2003.

[CNSD93] C. Cruz-Neira, D. Sandin, and T. DeFanti. Surround-screen projection-based virtual reality: The design and implementation of the cave. In *Proceedings of SIGGRAPH '93*, pages 135–142. ACM SIGGRAPH, 1993.

[COCS00] D. Cohen-Or, Y. Chrysanthou, and C. Silva. A survey of visibility for walkthrough applications. In *Proceedings of EUROGRAPHICS 2000, course notes*, 2000.

[COFHZ98a] D. Cohen-Or, G. Fibich, D. Halperin, and E. Zadicario. Conservative visibility and strong occlusion for view space partitioning of densely occluded scenes. *Computer Graphics Forum*, 17(3):243–254, 1998.

[COFHZ98b] D. Cohen-Or, G. Fibich, D. Haperin, and E. Zadicario. Conservative visibility and strong occlusion for viewspace partitioning of densely occluded scenes. In *Proceedings of EUROGRAPHICS 1998*, 1998.

[COI] Coin3d. http://www.coin3d.org.

[Con] Web3D Consortium. Vrml (x3d). http://www.web3D.org.

[Cor] R3vis Corporation. Openrm scene graph. http://www.openrm.org.

[COS94] D. Cohen-Or and Z. Sheffer. Proximity clouds - an acceleration technique for 3d grid traversal. *The visual computer*, 11:27–38, 1994.

[COZ98] D. Cohen-Or and E. Zadicario. Visibility streaming for network-based walkthroughs. *Graphics Interface '98*, pages 1–7, 1998.

[CSM95] T. Cassen, K.R. Subramanian, and Z. Michalewicz. Extending graphics hardware for occlusion queries in opengl. In *Proceedings of Graphics Interface '95*, pages 16–19, May 1995.

[CT96] S. Coorg and S. Teller. Temporally coherent conservative visibility. In *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, pages 78–87, 1996.

[CT97] S. Coorg and S. Teller. Real-time occlusion culling for models with large occluders. In *Symposium on Interactive 3D Graphics*, pages 83–90, 1997.

[DBHOS94] M. De Berg, D. Halperin, M. Overmars, and M. Snoeyink, Van Kreveld. Efficient ray shooting and hidden surface removal. *Algorithmica*, 12:30–53, 1994.

[DBVKOS97] M. De Berg, M. Van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer Verlag, Berlin Heidelberg, Germany, 1997.

[DLG90] N. Dyn, D. Levin, and J. Gregory. A butterfly subdivision scheme for surface interpolation with tension control. *ACM Transactions on Graphics*, 9(2):160–169, 1990.

[Dör01] R.J. Dörner. *Erstellung und Präsentation von Animationen für Trainingszwecke*. PhD thesis, Universität Frankfurt a.M., 2001.

[DVS01] A. D'Souza, S. Vijayakumar, and S. Schaal. Learning inverse kinematics. In *Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 298–303, 2001.

[EaSK96] J.L. Encarnação, W. Straßer, and R. Klein. *Graphische Datenverarbeitung I & II*. Oldenbourg Verlag, 1996.

[EaSL03] J.L. Encarnação, J. Sahm, and V. Luckas. Komponentenbasierte 3d animation in verteilten umgebungen. *Thema Forschung*, pages 58–64, 2003.

[EDD+95] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. Multiresolution analysis of arbitrary meshes. In *Proceedings of SIGGRAPH 1995*. ACM SIGGRAPH, 1995.

[EMBI01] C. Erikson, D. Manocha, and W.V. Baxter III. Hlods for faster display of large static and dynamic environments. *ACM Symposium on Interactive 3D Graphics*, 2001.

[Eri96] C.M. Erikson. Polygonal simplification: An overview. Technical Report TR96-016, Department of Computer Science, Chapel Hill, 1996.

[FA85] W.R. Franlin and V. Akman. Octree data structures and creation by stacking. In D. Magnenat-Thalmann and N. Thalmann, editors, *Computer generated images, state of the art*, pages 176–185, Tokyo, June 1985. Springer Verlag.

[FFH+99] F. Faure, C. Faisstnauer, G. Hesina, A. Aubel, J.C. Nebel, J.D. Gascuel, and F. Labrosse. Collaborative animation over the network. *Computer Animation '99*, pages 107–117, 1999.

[FI85] A. Fujimoto and K. Iwata. Accelerated ray tracing. In T.L. Kunii, editor, *Computer Graphics: Visual Technology and Art*, pages 41–65, 1985.

[FKN80] H. Fuchs, Z.M. Kedem, and B.F. Naylor. On visible surface generation by a priori tree structures. In *Proceedings of SIGGRAPH 1980*, pages 124–133. ACM SIGGRAPH, 1980.

[For72] A.R. Forrest. Interactive interpolation and approximation by bézier polynomials. *Computer Journal*, 15:71–79, 1972.

[FRE] freeglut. http://freeglut.sourceforge.net.

[FS93] T.A. Funkhouser and C. Séquin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *Proceedings of SIGGRAPH 1993*. ACM SIGGRAPH, 1993.

[FS98] E. Frécon and M. Stenius. Dive: A scaleable network architecture for distributed virtual environments. *Distributed Systems Engineering Journal*, 5:91–100, 1998.

[Fun95] T.A. Funkhouser. Ring: A client-server system for multi-user virtual environments. *Symposium on Interactive 3D Graphics*, pages 85–92, 1995.

[Fun96a] T.A. Funkhouser. Database management for interactive display of large architectural models. *Graphics Interface*, 1996.

[Fun96b] T.A. Funkhouser. Network topologies for scalable multi-user virtual environments. In *Proceedings of IEEE VRAIS '96*, pages 222–228, 1996.

[FvDF+93] J. Foley, A. van Dam, S. Feiner, J. Hugues, and R. Phillips. *Introduction to Computer Graphics*. Addison Wesley, 1993.

[GH97] M. Garland and P.S. Heckbert. Surface simplification using quadric error metrics. In *Proceedings of SIGGRAPH 1997*, pages 209–216. ACM SIGGRAPH, 1997.

[GH98] M. Garland and P. Heckbert. Simplifying surfaces with color and textures using quadric error metrics. *IEEE Visualization '98*, pages 263–269, 1998.

[GK94] N. Greene and M. Kass. Error-bounded antialiased rendering of complex environments. In *Proceedings of SIGGRAPH 1994*, pages 59–66. ACM SIGGRAPH, 1994.

[GKM93] N. Greene, M. Kass, and G. Miller. Hierachical z-buffer visibility. In *Proceedings of SIGGRAPH 1993*, pages 231–240. ACM SIGGRAPH, 1993.

[Gla84] A.S. Glassner. Space subdivision for fast ray tracing. *IEEE Computer Graphics and Applications*, pages 15–22, October 1984.

[Gla88] A.S. Glassner. Spacetime ray tracing for animation. *IEEE Computer Graphics and Applications*, 8(2):60–70, March 1988.

[GO97] J.E. Goodman and J. O'Rourke, editors. *Handbook of Discrete and Computational Geometry*. CRC Press LLC, 1997.

[GSPN95] S. Gurminder, L. Serra, W. Png, and H. Ng. Bricknet: Sharing object behaviors on the net. In *Proceedings of IEEE Virtual Reality Annual International Symposium*, pages 19–25, March 1995.

[Hac00] S. Hacker. *MP3: The Definitive Guide*. O'Reilly & Associates, Berlin Heidelberg, Germany, May 2000.

[Hag96] O. Hagsand. Interactive multiuser ves in the dive system. *IEEE Multimedia Magazine*, 3(1), 1996.

[HCB+99] P. Haffner, Y.L. Cun, L. Bottou, P. Howard, and P. Vincent. Color documents on the web with djvu. In *Proceedings of the International Conference on Image Processing*, pages 239–243, October 1999.

[HDDM93] H. Hoppe, T. DeRose, T. Duchamp, and W. McDonald, J.and Stuetzle. Mesh optimization. In *Proceedings of SIGGRAPH 1993*, pages 19–26. ACM SIGGRAPH, 1993.

[Hei04] A. Heizenreder. Evaluierung und entwicklung einer adaptiven repräsentation von areas of interest. Master's thesis, Studienarbeit, TU Darmstadt, 2004.

[HG95] P. Heckbert and M. Garland. Survey of polygonal surface simplification algorithms. Technical Report CMU-CS-95-194, Carnegie Mellon University, 1995.

[HL92] J. Hoschek and D. Lasser. *Grundlagen der geometrischen Datenverarbeitung*. Teubner, 1992.

[HM97] M. Harrison and M. Mclennan. *Effective Tcl/Tk Programming*. Addison Wesley, 1997.

[HMC+97] T. Hudson, D. Manocha, J. Cohen, M. Lin, K. Hoff, and H. Zhang. Accelerated occlusion culling using shadow frustra. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 1–10, 1997.

[Hop96] H. Hoppe. Progressive meshes. In *Proceedings of SIGGRAPH 1996*, pages 99–108. ACM SIGGRAPH, 1996.

[Hop97] H. Hoppe. View-dependent refinement of progressive meshes. In *Proceedings of SIGGRAPH 1997*, pages 189–198. ACM SIGGRAPH, 1997.

[Hop98] H Hoppe. Efficient implementation of progressive meshes. *Computer & Graphics*, 22(1):27–36, 1998.

[HPF] Hp-flag. http://www.opengl.org/documentation/specs/version1.2/HPspecs.

[HS98] G. Hesina and D. Schmalstieg. A network architecture for remote rendering. In *Proceedings of Second International Workshop on Distributed Interactive Simulation and Real-Time Applications*, pages 88–91, 1998.

[Hun78] G.M. Hunter. *Efficient Computation and Data Structures for Graphics*. PhD thesis, Department of Electrical Engineering and Computer Science, Princeton University, September 1978.

[Inc] Silicon Graphics Inc. Glut. http://www.sgi.com/software/opengl/glut.html.

[Jam99] A. James. *Binary space partitioning for accelerated hidden surface removal and rendering of static environments*. PhD thesis, University of East Anglia, August 1999.

[JPE] Jpeg 2000. http://www.jpeg.org/jpeg2000.

[JT80] C. Jackins and S.L. Tanimoto. Oct-trees and their use in representing three-dimensional objects. *CGIP*, 14(3):249–270, 1980.

[KAO] Kaon. http://www.kaon.com.

[Kaz93] R. Kazman. Making waves: On the design of architectures for low-end distributed virtual environments. In *Proceedings of IEEE Virtual Reality Annual International Symposium*, pages 443–449, 1993.

[KCCO00] V. Koltun, Y. Chrysanthou, and D. Cohen-Or. Virtual occluders: An efficient intermediate pvs representation. *Rendering Techniques 2000: 11th Eurographics Workshop on Rendering*, pages 59–70, 2000. June.

[KHM+98] J.T. Klosowski, M. Held, J.S.B. Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE Transactions Visualization Computer Graphics*, 4(1):21–36, January 1998.

[KK86] T.L. Kay and K.T. Kajiya. Ray tracing complex scenes. *Computer Graphics*, 20(4):269–278, March 1986.

[KKF+02] J. Klein, J. Krokowski, M. Fischer, M. Wand, R. Wanka, and F. Meyer auf der Heide. The randomized sample tree: A data structure for interactive walkthroughs in externally stored virtual environments. *VRST '02*, 2002.

[Kle97] R. Klein. Multiresolution representations for surfaces meshes. Technical report, Wilhelm-Schickard-Institut, GRIS, Universität Tübingen, Germany, 1997.

[KLI] Klimt. http://studierstube.org/klimt.

[KLS96] R. Klein, G. Liebich, and W. Strasser. Mesh reduction with error control. In *IEEE Visualization 1996 Conference Proceedings*, pages 311–318. IEEE, 1996.

[KMGL99] S. Kumar, D. Manocha, W. Garrett, and M. Lin. Hierarchical back-face computation. *Computer & Graphics*, 23(5):681–692, October 1999.

[KS97] L. Kobbelt and P. Schröder. Constructing variationally optimal curves through subdivision. Technical Report CS-TR-97-05, California Institute of Technology Computer Science Department, 1997.

[KS99] J.T. Klosowski and C.T. Silva. Rendering on a budget: A framework for time-critical rendering. *IEEE Visualization '99*, pages 125–122, October 1999.

[KS00] J.T. Klosowski and C.T. Silva. The prioritized-layered projection algorithm for visible set estimation. *IEEE Transactions on Visualization and Computer Graphics*, 6(2):108–123, 2000.

[LD98] V. Luckas and R. Dörner. Using object-oriented concepts for 3d visualization and validation of industrial scenarios. In R. Zobel and D. Möller, editors, *Proceedings of 12th European Simulation Multiconference 1998: Simulation - Past, Presence and Future*, pages 87–91, Manchester, UK, 1998. Society for Computer Simulation International (SCS), Arbeitsgemeinschaft Simulation (ASIM).

[LE97] D. Luebke and C. Erikson. View-dependent simplification of arbitrary polygonal environments. In *Proceedings of SIGGRAPH 1997*, pages 199–208. ACM SIGGRAPH, 1997.

[Lev97] M. Levoy. The digital michelangelo project: 3d scanning of large statues. In *Proceedings of SIGGRAPH 2000*, pages 131–144. ACM SIGGRAPH, 1997.

[LG95a] D. Luebke and C. Georges. Portals and mirrors: Simple, fast evaluation of potentially visible sets. *Symposium on Interactive 3D Graphics*, pages 105–106, 1995.

[LG95b] D. Luebke and C. Georges. Portals and mirrors: Simple, fast eveluation of potentially visible sets. In *Proceedings of the Symposium on Interactive 3D Graphics*, 1995.

[LHM+97] R. Lea, Y. Honda, K. Matsuda, O. Hagsand, and M. Stenius. Issues in the design of a scalable shared virtual environment for the internet. In *Proceedings of the HICSS '97*, 1997.

[Lin00] P. Lindstrom. Out-of-core simplification of large polygonal models. In *Proceedings of SIGGRAPH 2000*, pages 259–262. ACM SIGGRAPH, 2000.

[Loo87] C. Loop. Smooth subdivision surfaces based on triangles. Master's thesis, University of Utah, 1987.

[Lou94] M. Lounsbery. *Multiresolution Analysis for Surfaces of Arbitrary Topological Type*. PhD thesis, University of Washington, September 1994.

[LS01] P. Lindstrom and C.T. Silva. A memory insensitive technique for large model simplification. In *IEEE Visualization 2001 Conference Proceedings*. IEEE, 2001.

[LT97] K.L. Low and T.S. Tan. Model simplification using vertex-clustering. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics*, pages 75–82, 1997.

[Luc00] V. Luckas. *Elementbasierte, effiziente und schnelle Generierung von 3D Visualisierungen und 3D Animation*. PhD thesis, Technische Universität Darmstadt, Darmstadt, 2000.

[Mal89] S. Mallat. A theory for multiresolution signal decomposition: The wavelet representation. *IEEE transactions on Pattern Analysis and Machine Intelligence*, 1989.

[MB90] J.D. MacDonald and K.S. Booth. Heuristics for ray tracing using space subdivision. *The visual computer*, 6:153–166, 1990.

[MCO97] Y. Mann and D. Cohen-Or. Selective pixel transmission for navigation in remote virtual environments. *Computer Graphics Forum*, 16(3):201–206, 1997.

[Mea80] D. Meagher. Octree encoding: A new technique for the representation, manipulation, and display of arbitrary 3-d objects by computer. Technical Report IPL-TR-80-111, Image Processing Laboratory, Rensselaer Polytechnic Institute, 1980.

[MF96] B. MacIntyre and S. Feiner. Language level support for exploratory programming of distributed virtual environments. *Symposium on User Interface Software and Technology, ACM UIST '96*, 1996.

[MG01] T.B. Moeslund and E. Granum. A survey of computer vision based human motion capture. *Computer Vision and Image Understanding*, 81(3):231–268, March 2001.

[Mica] Microsoft. Direct 3d. http://www.microsoft.com/windows/directx.

[Micb] Sun Microsystems. Java3d. http://java.sun.com/products/java-media/3D.

[MS95] P.W.C Maciel and P. Shirley. Visual navigation of large environments using textured clusters. *ACM Symposium on Interactive 3D Graphics*, pages 95–102, 1995.

[Mur99] T.M. Murali. *Efficient Hidden-Surface Removal in Theory and in Praxis*. PhD thesis, Brown University, Providence, May 1999.

[MWJ97] J.P. Müller, M.J. Wooldridge, and N.R. Jennings, editors. *Intelligent Agents III. Agent Theories, Architectures, and Languages*. Springer, 1997.

[MZP+94] M. Macedonia, M. Zyda, D. Pratt, P. Barham, and S. Zesswitz. Npsnet: A network software architecture for large scale virtual environments. *Presence*, 3(4), 1994.

[Nay92] B.F. Naylor. Partitioning tree image representation and generation from 3d geometric models. In *Proceedings of Graphics Interface 1992*, pages 201–212, 1992.

[Net] Integrated Virtual Networks. Cybertown. http://www.cybertown.com.

[NNS72] M.E. Newell, R.G. Newell, and T.L. Sancha. A solution to the hidden surface problem. In *ACM Nat. Mtg.*, 1972.

[O2C] O2c. http://www.o2c.de.

[OPEa] Openscenegraph. http://openscenegraph.sourceforge.net/index.html.

[OPEb] Opensg. http://www.opensg.org.

[PE98] A. Puri and A. Eleftheriadis. Mpeg-4: An object-based multimedia coding standard supporting mobile applications. *ACM Mobile Networks and Applications Journal, Special Issue on Mobile Multimedia Communications*, 3(1):5–32, June 1998.

[PH97] J. Popovic and H. Hoppe. Progressive simplicial complexes. In *Proceedings of SIGGRAPH 1997*, pages 217–224. ACM SIGGRAPH, 1997.

[Pin88] L.A. Pineda. A compositional semantics for graphics. In *Proceedings of EUROGRAPHICS 1988*, pages 155–169, 1988.

[Pla93] H. Plantinga. Conservative visibility preprocessing for efficient walkthroughs of 3d scenes. In *Proceedings of Graphics Interface 1993*, pages 166–173, 1993.

[PRI] Prince of persia 3. http://www.prince-of-persia.com/tale03.

[Pri00] C. Prince. Progressive meshes for large models of arbitrary topology. Master's thesis, University of Washington, 2000.

[Qui00] H. Quin. Fem-based dynamic subdivision splines. In *Proceedings of the Eighth Pacific Conference on Computer Graphics and Applications*. IEEE, 2000.

[RAJ96] E. Reinhard, Kok A.J.F., and F.W. Jansen. Cost prediction in ray tracing. In P. Hanrahan and W. Purgathofer, editors, *Rendering Techniques '97*, pages 42–51, Porto, Portugal, 1996. Springer Verlag.

[RB93] J. Rossignac and P. Borrel. Multi-resolution 3d approximations for rendering complex scenes. *Computer Graphics: Methods and Applications*, pages 455–465, 1993.

[Ree81] W.T. Reeves. Inbetweening for computer animation utilizing moving point constraints. In *Proceedings of SIGGRAPH '81*, pages 263–269. ACM SIGGRAPH, 1981.

[Ros99] Jarek Rossignac. Edgebreaker: Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, 5(1):47–61, 1999.

[RR78] D. Reddy and S. Rubin. Representation of three-dimensional objects. Technical Report CMU-CS-78-113, Computer Science Department, Carnegie-Mellon University, 1978.

[SAG84] T.W. Sedelberg, D.-C. Anderson, and R.C. Goldman. Implicit representation of parametric curves and surfaces. *Computer Vision, Graphics and Image Processing*, 28:72–84, 1984.

[Sah98] J. Sahm. Komponentenbasierte 2d-animationselemente. Master's thesis, Technische Universität Darmstadt, 1998.

[Sam89] H. Samet. *The design and analysis of spatial data structures*. Addison Wesley, 3 edition, 1989.

[SBGS69] R. Schumacker, B. Brand, M. Gilliland, and W. Sharp. Study for applying computer-generated images to visual simulation. Technical Report Technical Report AFHRL-TR-69-14, NTIS AD700735, U.S. Air Force Human Resources Lab., Air Force Systems Command, Brooks AFB, TX, 1969.

[SBS03] J. Sahm, H. Birthelmer, and I. Soetebier. A client-server architecture for the cooperative visualization of large, interactive and dynamic 3d scenes. In *Proceedings of Simulation und Visualisierung 2003*, pages 387–403, 2003.

[SBSL04] I. Soetebier, I. Birthelmer, J. Sahm, and V. Luckas. Managing large progressive meshes. *Computer & Graphics*, 28(5):691–701, 2004.

[SC96] M. Slater and Y. Chrysanthou. View volume culling using a probabilistic caching scheme. In S. Wilbur and M. Bergamasco, editors, *Proceedings of Framework for Immersive Virtual Environments FIVE*, December 1996.

[Sch] D.C. Schmitt. Ace. http://www.cs.wustl.edu/ schmidt/ACE.html.

[Sch97] W.J. Schroeder. A topology modifying progressive decimation algorithm. In *IEEE Visualization 1997 Conference Proceedings*, pages 205–212. IEEE, 1997.

[SCH+01] L. Shou, J. Chionh, Z. Huang, Y. Ruan, and K. L. Tan. Walking through a very large virtual environment in real-time. In *International Conference on Very Large Data Bases*, pages 401–410, 2001.

[Sen] Sence8. Worldtoolkit. http://www.sense8.com.

[SG93] C. Shaw and M. Green. The mr toolkit peers package and experiment. In *Proceedings of IEEE Virtual Reality Annual International Symposium*, pages 463–469, September 1993.

[SG96] O. Sudarsky and C. Gotsman. Output-sensitive visibility algorithms for dynamic scenes with applications to virtual reality. In *Proceedings of EUROGRAPHICS 1996*, 1996.

[SG99] O. Sudarsky and C. Gotsman. Dynamic scene occlusion culling. *IEEE Transactions on Visualization and Computer Graphics*, 5(1):13–29, 1999.

[SGIa] SGI. Iris performer. http://www.sgi.com/software/performer.

[SGIb] SGI. Open inventor. http://www.sgi.com/software/inventor.

[SGIc] SGI. Opengl. http://www.opengl.org.

[SGId] SGI. Opengl vizserver. http://www.sgi.com/software/vizserver.

[SGLS93] C. Shaw, M. Green, J. Liang, and Y. Sun. Decoupled simulation in virtual reality with the mr toolkit. *ACM Transactions on Information Systems*, 11(3), 1993.

[SH74] I.E. Sutherland and G.W. Hodgman. Reentrant polygon clipping. *CACM*, 17(1):32–42, 1974.

[SLS+96] J. Shade, D. Lischinski, D.H. Salesin, T. DeRose, and J. Snyder. Hierarchical image caching for accelerated walkthroughs of complex environments. In *Computer Graphics Proceedings, Annual Conference Series*, pages 75–82, 1996.

[SM99] B. Schneider and I. Martin. An adaptive framework for 3d graphics over networks. *Computer & Graphics*, 23(6):867–874, 1999.

[Smi98] B. Smits. Efficiency issues for ray tracing. *Journal of Graphics Tools*, 3(2):1–14, 1998.

[Spa66] H. Spanier. *Algebraic Topology*. McGraw-Hill, New York, 1966.

[SS04] J. Sahm and I. Soetebier. A client-server-scenegraph for the cooperative visualization of large and dynamic 3d scenes. *Journal of The 12th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision 2004 (WSCG 2004)*, pages 387–394, 2004.

[SSB03] J. Sahm, I. Soetebier, and H. Birthelmer. Realtime visibility culling for the visualization and transmission of large 3d scenes. In *High Performance Computing Symposium 2003*, pages 145–154, Orlando, Florida, 2003.

[SSB04a] J. Sahm, I. Soetebier, and H. Birthelmer. Efficient representation and streaming of 3d scenes. *Computer & Graphics*, 28(1):15–24, 2004.

[SSB04b] I. Soetebier, J. Sahm, and H. Birthelmer. Client-server infrastructure for interactive 3d multi-user environments. In *Poster Proceedings of The 12th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision 2004 (WSCG 2004)*, pages 165–168, 2004.

[SSBB02] J. Sahm, I. Soetebier, H. Birthelmer, and H. Blechschmied. Wap for graphical objects, technology demonstration at the intergeo 2002 fair. *Computer Graphik Topics 2002*, 14(6), 2002.

[Ste97] A. Stewart. Hierarchical visibility in terrains. *Eurographics Rendering Workshop 1997*, pages 217–228, 1997.

[Sub90] T.M. Subramanian. *Adapting search structures to scene characteristics for ray tracing*. PhD thesis, University of Austin, Texas, December 1990.

[SVBL01] J. Sahm, J. Vollinga, E. Blechschmitt, and V. Luckas. Next generation geometry data exchange. In *International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet SSGRR*, pages 394–400, L'Aquila, Italy, 2001.

[SVNB99] C. Saona-Vazquez, I. Navazo, and P. Brunet. The visibility octree: A data structure for 3d navigation. *Computer & Graphics*, 23(5):635–644, 1999.

[SW94] D. Snowdon and A. West. The aviary vr-system. a prototype implementation. *6th ERCIM Workshop*, 1994.

[SZL92] W.J. Schroeder, J.A. Zarge, and W.E. Lorensen. Redecimation of triangle meshes. In *Proceedings of SIGGRAPH 1992*, pages 65–70. ACM SIGGRAPH, 1992.

[Tel92] S. Teller. *Visibility Computations in Densely Occluded Polyhedral Environments*. PhD thesis, Department of Computer Science, University of California, Berkeley, 1992.

[TL01] E. Teler and D. Lischinski. Streaming of complex 3d scenes for remote walkthroughs. In *Proceedings of EUROGRAPHICS 2001*, 2001.

[Tro] Trolltech. Qt. http://www.trolltech.com.

[TS91] S. Teller and C.H. Sequin. Visibility preprocessing for interactive walkthroughs. In *Proceedings of SIGGRAPH 1991*, pages 61–69. ACM SIGGRAPH, 1991.

[Tur92] G. Turk. Re-tiling polygonal surfaces. In *Proceedings of SIGGRAPH 1992*, pages 55–64. ACM SIGGRAPH, 1992.

[VM02] G. Varadhan and D. Manocha. Out-of-core rendering of massive geometric environments. *IEEE Visualization 2002*, pages 69–76, 2002.

[WA77] K. Weiler and K. Atherton. Hidden surface removal using polygon area sorting. *ACM Computer Graphics*, 11(2):214–222, July 1977.

[WAB+96] R. Waters, D. Anderson, J. Barrus, D. Brogan, M. Casey, S. McKeown, T. Nitta, I. Sterns, and W. Yerazunis. Diamondpark and spline: A social virtual reality system with 3d animation, spoken interaction and runtime modificability. Technical Report TR-96-02a, Mitsubishi Electronic Research Laboratory, 1996.

[War69] J. Warnock. A hidden-surface algorithm for computer generated half-tone pictures. Technical Report TR 4-15, NTIS AD-753 671, Department of Computer Science, University of Utah, 1969.

[Wat70] G.S. Watkins. A real-time visible surface algorithm. Technical Report UTECH-CSc-70-101, Department of Computer Science, University of Utah, Salt Lake City, Utah, 1970.

[WFP+01] M. Wand, M. Fischer, I. Peter, F. Meyer auf der Heide, and W. Straßer. The randomized z-buffer algorithm: Interactive rendering of highly complex scenes. In *Proceedings of SIGGRAPH 2001*, pages 361–370. ACM SIGGRAPH, 2001.

[WHG84] H. Weghorst, G. Hooper, and D.P. Greenberg. Improved computational methods for ray tracing. *ACM Transactions on Graphics*, 3(1):52–69, January 1984.

[WNDS99] M. Woo, J. Neider, T. Davis, and D. Shreiner. *The OpenGL(R) Programming Guide: The Official Guide to Learning OpenGL.* Addison Wesley, 3 edition, 1999.

[WOR] Worlds.com 3d portal. http://www.worlds.net.

[WPF90] A. Woo, P. Poulin, and A. Fourier. A survey of shadow algorithms. *IEEE Computer Graphics and Applications*, 10(6):13–31, 1990.

[WS99] P. Wonka and D. Schmalstieg. Occluder shadows for fast walkthroughs of urban environments. *Computer Graphics Forum*, pages 51–60, 1999.

[WSC+95] K.Y. Whang, J.W. Song, J.W. Chang, J.Y. Kim, W.S. Choand, C.M. Park, and I.Y. Song. Octree-r: An adaptive octree for efficient ray tracing. *IEEE Transactions on Visualization and Computer Graphics*, 1:343–349, 1995.

[WWS00] P. Wonka, M. Wimmner, and D. Schmalstieg. Visibility preprocessing with occluder fusion for urban walkthroughs. *Rendering Techniques 2000: 11th Eurographics Workshop on Rendering*, pages 71–80, 2000. June.

[WZ98] K. Watsen and M. Zyda. Bamboo - supporting dynamic protocols for virtual environments. *Image Conference*, 1998.

[XV96] J.C. Xia and A. Varshney. Dynamic view-dependent simplification for polygonal models. In *IEEE Visualization 1996 Conference Proceedings*, pages 335–344. IEEE, 1996.

[Zha98] H. Zhang. *Effective Occlusion Culling for the Interactive Display of Arbitrary Models.* PhD thesis, Department of Computer Science, Chapel Hill, 1998.

[ZIP] zlib. http://www.gzip.org/zlib/.

[ZMHHI97] H. Zhang, D. Manocha, T. Hudson, and K.E. Hoff III. Visibility culling using hierarchical occlusion maps. In *Proceedings of SIGGRAPH 1997*, pages 77–88. ACM SIGGRAPH, 1997.

[ZPF+93] M.J. Zyda, R.P. Pratt, J.S. Falby, C. Lombardo, and K.M. Kelleher. The software required for the computer generation of virtual environments. *Presence*, 2(2):130–140, March 1993.

Tabellenverzeichnis

Abbildungsverzeichnis

Anhang A

Betreute Diplom-, Studien- und Bachelorarbeiten

A.1 2001

- C. Drossel: *Verteilte, interaktive und agentenbasierte 3D-Animation.* Technische Universität Darmstadt (TUD), Diplomarbeit Fachbereich Informatik (FB 20), Fachgebiet Graphisch-Interaktive Systeme (GRIS), Darmstadt, 2001.

- H. Birthelmer: *Evaluation und Implementierung von Verfahren zur Geometriereduktion mit Fehlerschranken.* Diplomarbeit Fachhochschule Darmstadt (FH), Fachbereich Informatik, 2001.

A.2 2002

- S. Miksch: *Hierarchische Datenstrukturen zur Darstellung dynamischer 3D Szenen.* Bachelorarbeit Fachhochschule Darmstadt (FH), Fachbereich Informatik, Darmstadt, 2002.

- S. Barthel, S. Böhm: *Konzeption und Realisierung eines Designtools für 3D Szenen.* Bachelorarbeit Fachhochschule Darmstadt (FH), Fachbereich Informatik, Darmstadt, 2002.

A.3 2004

- A. Heizenreder: *Evaluierung und Entwicklung einer adaptiven Repräsentation von Areas of Interest.* Studienarbeit Fachbereich Informatik (FB 20), Fachgebiet Graphisch-Interaktive Systeme (GRIS), Darmstadt, 2004.

- S. Schäfer: *Effektive räumliche Sortierung großer, dynamischer 3D Szenen*. Diplomarbeit Fachbereich Informatik (FB 20), Fachgebiet Graphisch-Interaktive Systeme (GRIS), Darmstadt, 2004.

Anhang B

Lebenslauf

Name:	Jörg Sahm	
Geburtstag:	6. Dezember 1971	
Geburtsort:	Offenbach a.M.	
Schulbildung:	1978 – 1982	Grundschule, Ernst Reuter Schule Offenbach a.M.
	1982 – 1984	Förderstufe, Bachschule Offenbach a.M.
	1984 – 1991	Gymnasium, Leibnizschule Offenbach a.M. (Abschluss Abitur)
Wehrdienst:	Oktober 1991 – Oktober 1992	
Studium:	Oktober 1992 – Oktober 1998	Informatik, an der Technischen Universität Darmstadt (TUD), (Abschluss Diplom-Informatiker)
Berufspraxis:	1. Januar 1999 – 30. Juni 1999	Wissenschaftliche Hilfskraft mit Abschluss, Institut für Graphische Datenverarbeitung (IGD), Abteilung Animation & Bildkommunikation (A3), Darmstadt
	1. Juli 1999 – 31. März 2000	Wissenschaftliche Mitarbeiter, Fraunhofer Institut für Graphische Datenverarbeitung (IGD), Abteilung Animation & Bildkommunikation (A3), Darmstadt

1. April 2000 – 31.12.2003	Wissenschaftlicher Mitarbeiter, Graphisch Interaktive Systeme (GRIS), Abteilung Animation & Bildkommunikation (A3), Darmstadt
Seit 1. Januar 2004	Wissenschaftliche Mitarbeiter, Fraunhofer Institut für Graphische Datenverarbeitung (IGD), Abteilung Animation & Bildkommunikation (A3), Darmstadt